An expandable input/output and graphics system for distributed memory parallel computers

Hafeez, Muhammad

Award date:
1990

Awarding institution:
University of Bath

AN EXPANDABLE INPUT/OUTPUT AND GRAPHICS SYSTEM FOR DISTRIBUTED MEMORY PARALLEL COMPUTERS

Submitted by Muhammad Hafeez,
M.Sc.(Phy), M.Sc.(Eng)
for the degree of
Doctor of Philosophy
of the University of Bath
1990

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Summary

The implementation of an Input/Output system and a graphics system for parallel processors is described. Both systems are designed around currently available VLSI circuits. The hardware was designed to provide parallelism at the service device level. Both systems can be expanded in parallel to achieve higher throughput. This parallel expandability of the individual systems is also described.

Acknowledgements

The author is indebted to his supervisors, Mr A.R. Daniels, and Dr R.W. Dunn for their help and guidance throughout this work. He would also like to express his gratitude to Mr V.S. Gott for his continuous support with the development of hardware.

He would like to thank Professor J.F. Eastham, Head of the Power and Control Group and Head of School for provision of facilities at the School of Electrical engineering University of Bath.

Financial grant from the Government of Pakistan, Ministry of Science and Technology and leave for studies by the National Institute of Electronics Islamabad, Pakistan is gratefully acknowledged.

In addition, I would like to acknowledge the technical help of Philips Components Ltd. throughout this project.

At the end, the author would like to express his thanks to his colleagues at the University, in particular Mr C.G. Selwyn and Dr T. Berry for their help and support throughout this project and also during preparation of the thesis. He would also like to thank his wife for her support during studies and help in typing out the manuscript.

This thesis was produced using LaTeX running under the Helios operating system.

Xmu.h Xmu.o

Contents

List of principal symbols

ADn	MemADn, memory address/data bus (T800).
An	Address bus bit n.
ASN	Address strobe.
ATN	Attention signal on SCSI bus.
BCD	Binary Coded Decimal.
BCR	Border Colour Register.
BGACKN	Bus Grant Acknowledge.
BGN	Bus Grant.
BRN	Bus Request.
BSY	Busy, SCSI bus signal.
C/D	Control/Data signal on SCSI bus.
CASN	Column Address Strobe for DRAM.
CDA	Combined Disk Arrangement.
CLI	Command Line Interpreter.
CPU	Central Processing Unit.
CSION	VSC decoded Chip Select signal for I/O area.
CSR	Control and Status Register in VSC.
CSYNCN	Composite synchronising signal.
CYACKN	Cycle Acknowledge.
CYREQN	Cycle Request.
DCA	Dynamic Control Area (VSC).
DCB	Device Control Block.
DCR	Display Control Register (VSC).
Devinfo	Device Info structure in Tripos.
Devtab	Devices table in Tripos.
DMA	Direct Memory Access controller.
DPLL	Disk Phase Lock Loop.
DRAM	Dynamic Random Access Memory.
DRAMIO	VSC controlled DRAM access type I/Os.
DSCR	Device Status and Control Register.
DTACKN	Data Acknowledge signal.
ECA	Event Control Area (VSC).
ECAP	ECA Pointer.
EnVec	Environment Vector (for Tripos device).
FDC	Floppy Disk Controller.
FIFO	First In First Out.
FM	Frequency Modulation.
FPU	Floating Point Unit.
GaAs	Gallium Arsenide.
HALTN	Halt input pin (SCC68070).

HSYNCN	Horizontal synchronising signal.
I/O	Input/Output.
I2C	Inter-Integrated Circuit bus.
ICA	Image Control Area (VSC).
ID	Identity number.
IMDC	Intelligent Multiple Disk Controller.
ITB	Interrupt Transfer Block.
K-QPkt	Kernel to queue a packet onto a WorkQ.
LIR	Latched Interrupt Register.
LnIN	Link input signals (T800).
LnOUT	Link output signals (T800).
LS,LnS	Linkspecial and Linknspecial signals (T800). for link speed selection.
LSB	Least Significant Bit.
MFM	Modified Frequency Modulation.
MIMD	Multiple Instruction Multiple Data.
MMU	Memory Management Unit.
MRD	notMemRd signal (T800).
MRF	notMemRef signal (T800).
MSB	Most Significant Bit.
MSG	Message signal on SCSI bus.
MSn	notMemSn signals (T800).
MWBn	notMemWrBn signals (T800).
PICRn	Peripheral Interrupt Control Registers.
PLD	Programmable Logic Device.
PLL	Phase Lock Loop.
RAM	Random Access Memory.
RASN	Row Address Strobe.
RDYN	Ready input signal for DMA controlled data transfers.
RGB	Red, Green and Blue signals for video monitor.
RISC	Reduced Instruction Set Computer.
ROM	Read Only Memory.
RST	Reset signal on SCSI bus.
RSTINN	Reset input to VSC.
RSTOUTN	Reset output from VSC.
R/WN	Read Write signal.
RWorkQ	Receiver Work Queue.
SASI	Shugart Associates System Interface.
SCL	Serial CLock.
SCSI	Small Computer System Interface.
SDA	Serial Data.
SEL	Select signal on SCSI bus.
SIMD	Single Instruction Multiple Data.

SLPG	Screen Level Parallel Graphics.
SSP	Supervisor Stack Pointer.
TAS	Test And Set.
TCB	Task Control Block.
Tripos	Tripos operating system.
TWorkQ	Transmitter Work Queue.
UART	Universal Asynchronous Receiver Transmitter.
UASN	Upper Address Strobe.
UDSN	Upper Data Strobe.
USP	User Stack Pointer.
VAn	Video memory address bus signal.
VCO	Voltage Controlled Oscillator.
VDn	Video memory data bus signal.
VLSI	Very Large Scale Integrated circuits.
VSC	Video and System Controller (SCN66470).
VSYNCN	Vertical synchronising signal.
XCK1	External clock input pin on SCC68070.

Chapter 1

Introduction

The development of the microprocessor and its subsequent miniaturization and application has had an obvious impact on technology as well as society. The microprocessor, today, is being used in applications only remotely thinkable a few decades ago. From necessities to luxuries: numerous home appliances, robots, telephone systems, satellites, automobiles, word processors, personal computers, calculators, medical instruments and even within modern television sets, many are fitted with a small microcontroller. The microprocessors have changed the whole lifestyle of the modern world.

From the beginning of the development of microprocessors, development scientists have devoted significant efforts towards increasing the processing speed. In order to increase the speed of microprocessor based systems, faster devices using more modern technologies have been developed. This has resulted in reducing the size of the components to minimize the propagation delays between gates, with a large number of gates placed on a single chip.These higher speed devices tend to have a greater gate density.

The miniaturisation of the components has also led to very large scale integrated

circuits. Microprocessors with large data sizes [1] and several on-chip peripherals [2] are now available. This reduces both hardware design time and the cost of production. Faster random access memory (RAM) devices, both static and dynamic, are becoming available. This has also assisted the higher speed achievements of complete computing system. Further, techniques such as cache and memory paging [3,4] are used to reduce the memory speed constraints.

New types of technologies have also been exploited. Gallium Arsenide (GaAs) devices have been designed [5]. Even the earliest versions of the GaAs based microprocessors are more than twenty times faster than the normally available silicon based microprocessors [6]. Optical computers are also being developed [7,8]. These should work even faster.

1.1 Parallel processing

The optical computers is a thing of the future though not very far away. GaAs based microprocessors are available and are put into work with the hope of achieving real-time computing speed for complex programs [9]. A second approach more commonly exploited to achieve faster speeds is parallel processing. Programs are broken down into a number of smaller tasks and each processor is allocated to one of these small tasks to perform the whole operation in parallel. The ideal speed would be a system containing 'n' microprocessors working 'n' times faster than a system comprising just a single microprocessor. Task communication overheads and communication system contention, however, are the major prohibition in this context [10].

Real-time requirements can drive a computing system to the limits of the mem-

ory and CPU resources. The CPU and memory systems, therefore, involve a compromise between efficiency, response time, overheads, processing time and memory utilization [11]. Using more than one microprocessor not only gives the capability of increased speed, but it can also increase the reliability and provide for tolerance to processor failures.

Parallel processing can be classified [12] as single or multiple instruction streams with single or multiple data streams. Until recently the most commonly used systems were Single Instruction Multiple Data (SIMD) types, e.g. vector and array processors. However, Multiple Instruction Multiple Data (MIMD), i.e. multi-processor systems are now more common. These can be further classified as either loosely coupled communicating by messages, or tightly coupled communicating by a shared resource e.g memory. Loosely coupled systems have the advantage of being reconfigurable as and when required by a user. This makes them more adaptable for high speed computers and simulators.

1.2 Computer resources

One of the basic features of a computer is its ability to send or receive data to and from other devices and the outside world. This communication ranges from slow devices which send one character at a time, for example teletype device, to very fast devices which require a whole block of data to be transferred. Examples include a magnetic disk device or even another microprocessor or computer (if the machine is a part of a distributed network). The choice of the type of communication technology depends on the distance of the peripheral from the computer as well as the communication speed. The fast devices, close to the microcomputer are most efficient using a parallel data transfer mechanism, while

the slower devices can use a serial communication method.

The main data storage device is an important computer resource, handled by the memory manager. The complexity of the memory manager and the storage device depends on the operating system requirements [13]. Advanced operating systems support the concept of 'virtual memory' [14,15]. This concept allows the user to transparently extend the main memory (i.e. RAM) to a secondary storage device such as a hard disk unit. In these systems the main memory required to run a program is much smaller than the total memory required to store it. The memory manager moves segments of the running program from the secondary storage device to the main silicon storage as and when required. This data movement is transparent to the user, and the user views the storage device as a continuous address space spanning the whole of the processor address range.

This idea of virtual memory is not generally applied to parallel processing because, due to the higher costs involved, it is not normally possible to support each processor in the system with a secondary storage device. However, provision can be made to share the central secondary storage among all the processors executing a task [16]. The memory manager can provide this facility with an independent I/O controller managing the secondary storage. However, the speed of the storage device imposes a major bottleneck in this arrangement. This becomes more obvious with increasing numbers of processing elements. Another simpler arrangement would be to provide every microprocessor with enough main silicon storage.

1.3 Parallel graphics

Graphics has played a key role in allowing people to express their thoughts. As a presentation, design and expression tool, the computer is replacing traditional media. Sophisticated simulators are being used for training and analysis purposes to improve safety or reduce costs. One of the most common and well known example is the flight simulator. This type of applications require high speed computers to match the graphics output to the real- time requirements.

Two of the most basic functions of a graphics system are generating and refreshing images on the screen. Information relating to the image is stored in a 'graphics memory'. This memory is also sometimes referred as the 'bitmap memory'. The picture elements (pixels) in monochrome pictures can be represented by individual bits. Alternatively, colour pictures, depending upon the colour depth, require a group of bits for each pixel. In the 'bitmap memory', the lines and arcs from a display list are transformed and placed as a series of individual dots or pixels. This activation of dots is achieved through some scan conversion algorithm. This algorithm is repeated again and again to create the whole image. At the same time, the image memory must be updated or 'repainted' at least some 30 times per second on the screen to produce real-time visual effects.

The pixels in the graphic memory are usually neither complete words nor bytes. They tend to be produced by mathematical and logical operations. Integer addition, multiplication and logical operations are performed on the particular bits which constitute each pixel. This requires a quite extensive amount of computation.

As mentioned before, a high processing speed can be obtained by processing

data in parallel. The availability of dedicated graphic chips [17] has eased the processing load on the computer to some extent but still the picture repainting presents a significant bottle neck. Often, the image data requires a large amount of storage and this can exceed the limits of the main memory. This results in a lot of overhead time for transferring data between the main memory and some secondary storage device, e.g. the magnetic disk. Also, the data movement from the main memory to the 'bitmap memory' demands a high memory bandwidth.

1.4 About the thesis

Parallel processing offers a cheap solution for high speed computing. This has two major drawbacks. The first is the limitation of the secondary memory device (the magnetic disk). This does not offer enough bandwidth, i.e. the speed of data movement, to incorporate the idea of virtual memory for all processors. Unfortunately, high speed electronics can offer no solution for this problem. The slower mechanical interface is always a hindrance and it can only be solved by putting a number of disks in parallel, probably one disk and a controller with each participating processing node.

The second difficulty arises when updating the picture memory for graphics applications. Real-time solid modeling graphics demands a high memory bandwidth. The picture data can be processed with adequate speed using the parallel processing arrangement. But, even moving the processed data to the 'graphics memory' does not fit within the available memory bandwidth. The dynamic memories used for the 'bitmap memory' also require frequent refreshing. This reduces the memory bandwidth available for memory update and hence slows down the microprocessor. Static memories, on the other hand, consume more power and as a

result generate a lot more heat. These static memories are also expensive, have a larger number of connection pins per package and hence require more printed circuit board area.

This thesis presents the work carried out to find a solution to the above problem. One of the objectives of the work was to build one universal board, the graphics board, which could be used with a transputer based computer. The circuit was built using a minimum possible number of chips to increase the overall system reliability. This also eased the printed circuit board design and the construction of the boards.

The thesis discusses the two parts of the project. Firstly, details about the design of the I/O board are given. Before the discussion of the design of this board, a survey is given (chapter 2) to assess the requirements of a parallel processing system. A brief note about the hardware and the parallel device services is included. This is followed by chapter 3, in which the functions of the SCC68070, the main I/O processor and the Video and System Controller chip (VSC) are described. Chapters 4 to 6 are used to describe the design of the I/O board.

The peripherals attached to the I/O board include a console, one or more hard disks, floppy disks, a day/date clock, and a centronics printer. The hardware design of the floppy disk controller interface circuit is described in chapter 4. In chapter 5, a discussion about the circuits used to interface with the hard disk, the clock, and the printer is included. The structure of the device drivers is given in chapter 6 and the bootstrap design is included in the same chapter to complete the I/O board.

The T800 transputer [18] is introduced in chapter 7. This was used as a graphics

coprocessor for the SCC68070 and VSC to generate the pixel data for the 'graphics memory' which is controlled by the VSC. The T800 was used because of its built in capability for constructing parallel processing systems. The T800 coprocessor was designed as a stand alone processor with its own memory which could update the picture from other transputers in the system. Chapter 7 is dedicated to discussing this. In chapter 8, an effort is made to explain the operation of the parallel graphics system. The proposed display file and the related Image Control Area and the Dynamic Control Area (ICA and DCA) are given. Also, the interface to a colour palette is described. Chapters 9 and 10 conclude the thesis with the summary in chapter 9 and some suggestions in chapter 10 for further work to expand the system.

Chapter 2

A survey of parallel processing requirements

2.1 Introduction

Parallel processing is not a new idea in the fast changing computer world. The data processing markets and the military community had realised the need for faster data manipulators from the beginning and the researchers have always come up with innovative ideas. But it was only the advancement of Very Large Scale Integrated (VLSI) circuit technology and the reduced cost of the microcomputer components that pulled the computer users into this field. Ever reducing costs of memory devices and other peripherals has also encouraged the system designers to make even bigger and faster machines, thus realising 'real-time' processing speeds.

In parallel processing, modules of a single program or a number of individual programs are loaded onto different processors and are executed concurrently for some length of time. The objective is to reduce the queuing times for the user and to reduce the total running time for the program. Program modules can be designed to fit to the specific hardware and the application requirements. The

hardware can be made modular. The idea is to increase the computing power of the system by simply adding more microprocessors and dividing the programs into smaller modules to make use of the added processors. This can be cheaper than designing machines with differing performances for application specific purposes.

The processing speed is governed by the system architecture as well as the microprocessor architecture. For example, if a program does a complicated floating point computation, it would be better to use a processor which is designed for floating point operations. Similarly, if the program has to handle extensive symbol manipulations, it would better to use a microprocessor with a Central Processing Unit (CPU) with an instruction set which could perform these operations efficiently. The most modern VLSI microprocessors have a built in floating point unit (FPU) [18,19]. This combines both the instruction manipulator and the floating point unit into a single entity. Such devices make it easier to exploit the required type of operation (i.e. CPU or FPU) at the required moment.

When the different modules of a single program are running on different processors they are frequently required to communicate with each other. This interprocessor communication has an overhead in the form of some extra processing time in addition to the required data transfer time [20]. It would be ideal if the communication costs were zero. But, unfortunately, this is not the case. This overhead is related to the system architecture. In loosely coupled machines, the communication can require reconfiguration of the available communication buses within the system. As the number of processors increases, this system will require more buses to maintain the same communication level between processors. If the number of buses is limited, then, a communication 'bottleneck' can occur which will reduce the efficiency of a parallel processing program.

2.2 Parallel processors

The processing nodes in a parallel processing system have physical and logical links between each other and to the peripherals. The physical links are provided by the electronic circuits, cables and connectors in the backplane. These physical links are controlled and maintained by the software, thus establishing the logical channels for data transfers. A single processing element could have more than one channel [18]. In loosely coupled machines, the processors can establish a logical link to one of the several physical links (if more than one physical link is provided to each processor) by multiplexing them under the software control. In some tightly or closely coupled machines, where the processors are both physically and logically connected to their immediate neighboring counterparts (e.g. array processors), the data transfers can occur approximately with the speed of memory transfers. This helps to minimise the communication overheads.

2.2.1 Parallel processor system architectures

Parallel processors may have either fully interconnected architectures or they may be partially interconnected. The fully interconnected system involves a completely point to point interconnected multi-microcomputer system (Fig 2.1). In this type of the system the interconnect topology becomes very complex when many processors are added to the system. The interconnection of a large number of processing nodes would become expensive and the addition of every new processor would require a generic change in the system hardware. The system would hardly be a modular one. Addition of a new node would mean that one more interface controller would be added to each of the processors in the system, making the number of the interface controllers for each individual processor equal

to 'n-1' for an 'n' node system.

The fully interconnected architecture can be very useful for smaller systems. These systems can be relatively reliable, invoking minimum adverse effects in the event of node failures. In parallel processing systems with a larger number of processing nodes, however, it is more usual for a partial interconnect mechanism to be used. The basic interconnect mechanisms in the Multiple Instruction Multiple Data (MIMD) machines are shared memory, shared bus, star, ring, tree or hyper-cube architectures [21]. In a shared memory system, the shared memory can be the main memory (RAM) or the auxiliary memory (hard disk) or both. The processors have comparable capabilities and interact by shared access to the memory. The operating system provides the mechanisms to control this resource sharing. The data transfer requirements can be very tight requiring a high bus bandwidth [22]. Sometimes a multi-ported memory system can help to achieve fast memory accesses. Another solution would be to provide an independent memory module to each of the processing nodes in the form of cache. Alternatively, for a cluster of nodes, a similar method can be applied to provide the main memory. The size of this independent memory would depend upon the bus bandwidth, the communication rate and the type of applications the computer is intended for.

A shared communication bus would imply that the microprocessors can communicate over a common bus (fig 2.2). A central bus arbitration mechanism receives the requests for bus accesses from the communicating processors. The bus is granted either through hardware (e.g. daisy chaining) or through some software mechanism designed on some bus allocation algorithm (e.g. a preallocated priority or work sharing basis). The processors usually have their own exclusive memory units. The performance of the system depends on the bus bandwidth,

number of the processing nodes using the bus, the bus access mechanism and the average and peak message traffic rates.

If the physical length of the bus cable and the number of processors and couplers attached to the bus are kept to a minimum, the bus reliability can be increased. Similarly, the reliability is further increased by reducing the width of the bus. A single shielded cable bus is more reliable than multi-wire twisted pairs or a ribbon cable over a long distance. However, a bus with reduced width would be much slower compared to a multi-wire parallel bus. Occasionally, the system is provided with more than one global bus. This eases the bus bandwidth problem [23,24] and makes the communication faster.

In the star configuration, one processor is at the centre and acts as a system controller. It acts as a 'master' distributing the tasks to the other processing nodes and at the same time handling the Input/Output (I/O). The other processing nodes are attached to the 'master' with separate buses (Fig 2.3). For the inter-processor communication, the master would act as a multiplexing switch. Typically, when processor 'A' wants to talk to processor 'B', it requests the master to establish for it a message path to 'B'. If 'B' is ready to receive, the path is established and the communication starts. Meanwhile, if two other different nodes want a data interchange they can go ahead as long as 'A' and 'B' are not interfered with. But if they want to talk to 'A' or 'B', they will have to wait until the data interchange between 'A' and 'B' is complete and the path is released. The system is limited by the maximum number of message paths the master can provide at any one time. This in turn limits the communication bandwidth.

A basic ring multi-processor communication system consists of a high speed unidirectional digital communication channel arranged in a closed loop (fig 2.4).

These rings are further combined in parallel to implement the bidirectional data communication (fig 2.4). The microcomputers attached to the ring are provided with the hardware to interface them with the loop. The message from one node to the other is entered into the loop by the message initiating node. This message, then, circulates into the loop until it finds the required destination or goes back to the initiator. Once started, the message is removed from circulation either by the destination processor or by the initiator depending upon the protocol procedure. The ring interface hardware is intelligent enough to recognise whether the message is to be accepted, or relayed on to the next node. This message recognition does not involve the processor and therefore offers efficient message passing mechanism.

The packets of data, which are called 'frames' or 'slots', are marked with a destination and move around in the ring. Each ring interface controller receives the packet, checks the destination and relays the packet onto the next interface if it is not the destination. When the packet reaches the destination, the interface controller removes it from the ring and delivers it to the local processor. The disadvantage of a unidirectional communication loop is that if one of the interfaces failed to relay the message to the next interface, the next node would never know that there was a message in circulation. Because of these single point failures, the ring architecture was modified to a bidirectional communication system to introduce the fault tolerance. As each message is being received and transmitted by each of the interface controllers (i.e. multiple transmission and receptions are involved), it is prone to noise errors and can get more easily distorted compared to the shared bus point to point communication. If the packet is received by the correct destination then it can be easily found whether the received data is correct or corrupted by applying a parity check or some other error checking method. Moreover, a distortion in the destination address could mean that ei-

ther the packet is delivered to the wrong destination or if the destination address does not match to any of the system nodes, the 'lost packet' could remain in circulation, unnecessarily occupying a slot, until it is removed by the supervisor. Sometimes diagnostic packets are kept in circulation by the ring supervisor to make sure that the ring is functional. In the case of an interface failure, certain 'self-healing' techniques [25] can be applied to keep the loop functional.

In the tree structure [26], (Fig 2.5), the top level processor handles the task allocation process. It establishes the communication paths between different branches of the tree, and carries out supervisory tasks to make sure that the low level processors are 'healthy'. In this system, the information is passed vertically from top level of the tree to the processors at the bottom level. This information may consist of programs, data or commands, or a combination of all of these. The processors at the bottom of the tree are usually given the low level repetitive jobs while the command control and data processing are performed at the higher level. The data backup and store management is also done at the high level.

Depending upon the application, the processors at the higher levels of the tree might possess more communication and control capabilities. The communication between different branches of the tree is handled by the higher level processors. The transfer of a significant amount of data across more than one level can become prohibitive in terms of communication overhead. To avoid this, care must be taken while organising the program modules and distributing them to the processors so that the minimum possible communication involving more than one tree level occurs.

For a hyper-cube architecture, the microprocessors are organised in a multi dimensional structure [27]. A normal hyper-cube would require a minimum of eight

processors to connect them in the form of a three dimensional cube (fig 2.6). An 'n' dimensional cube could consist of 2^n processors (where n is an integer), with two processors at the end of each vertex of the cube. The communication distance between the processors is measured in terms of the processing nodes in the path of two communicating nodes. For example, in a three dimensional hyper-cube, the nodes at any edge can be regarded as near neighbours as they can communicate directly to each other. On the other hand, the processing nodes at the longest communication distance are at the diagonal edges of the cube as the data transfers between them would involve two other nodes as well. A further modification of the hyper-cube architecture is a double-bus hyper-cube (fig 2.7). This double-bus hyper-cube offers the same properties as the normal hyper-cube with the ability of connecting a larger number of processing nodes using a small number of buses.

Most MIMD applications involve a hybrid architecture exploiting positive points of each of the possible basic architectures. For a large distributed network, the hybrid architecture could mean that different nodes are connected into clusters in some basic form (e.g. hyper-cube) with further inter connection of these clusters by means of multiple buses. Optical fibre buses could be used as these offer much more bandwidth over a greater physical distance when compared to conventional multiple conductor buses [28]. Moreover, as optical fibres are not susceptible to electrical noise, they can be more reliable and hence can prove to be more efficient.

2.2.2 Inter-processor communication

As mentioned above, in a parallel processor system there are several processors, each of them executing code concurrently with the other processors. The code

loaded on an individual processor can be part of a common code or program (single user, multi-processor system) or it can be from the code for individual users (multi-user, multi-processor system). The processors involved in either of these systems frequently require an information interchange. The frequency of this information interchange depends on the type of the tasks and the way they are organised. Some architectures of parallel computers have hardware connection topology dependency in terms of inter-processor communication. In these types of machines, the tasks which require more frequent interaction are normally loaded on the near neighbours to facilitate a direct communication. Those tasks which require less frequent or no communication at all can be kept on more distant processors. These program modules are prepared by the programmers using prior knowledge and experience together with the support from the operating system. In distributed memory systems, the operating system is usually executed separately and independently on each processor.

In a multi-user, multi-processor system, usually one processor is allocated to one user to run their program. The inter-processor interaction in this type of system is normally between the processors and the I/O devices. In this case, the program is organised serially and loaded on the processor as one module. The system generally has one or more central storage devices (hard disks) and the processors have their own independent RAM devices. The users are connected to the processors either through some central I/O protocol mechanism or each user may have an exclusive connection to the designated processor. Because the user input devices (e.g. console) are normally connected to the machines through some serial bus (e.g RS 232-C) which is comparatively slower than the parallel bus, it may not matter whether the interaction is through some central I/O mechanism or if it is an exclusive one. However, for running the task, the processor would require utility support from devices such as the central hard disk unit. The hard

disk controller could be configured either to be accessible by a single 'master' processor, or it may be left to the individual processors to reconfigure the hard disk controller and make the data transfers by themselves.

In the first case, for faster data manipulation, the master has to serve the frequent slave requests as quickly as possible. The system would be dependent on one individual processor for the frequent disk requests and will be more vulnerable to failures. On the other hand this would require less complicated device handling routine and mutual exclusion mechanisms. In the second case, where the disk controller is reconfigurable and accessible by all the participating processing nodes, the system would be more fault tolerant. If the console and other peripheral handling is also done by the individual themselves themselves then the failure of any one processing node will have a minimal adverse effect on the system performance.

Single-user multi-processor systems are used for more advanced applications by more skilled and more well-versed computer users. The programs are written as individual modules and the programmer allows the modules to interact with each other occasionally to implement the whole operation successfully. While scheduling the tasks using prior knowledge, the more frequently interacting tasks are loaded on the near neighbour processors, thus minimising the communication distance. The inter-processor interaction in this type of application is more frequent between the processors than the I/O devices. The processors involved, if they have enough separate independent memory, could be loaded with the support software (i.e. utilities and other required subroutines) before they are set to run the actual task. In this way the processors could be organised to run the task with a minimum of hard disk support.

The processor interaction can also be required to implement the concurrency between the processors accessing a common resource. A 'test and set' semaphore is often used for this purpose. The processor which requires access to the resource, would first access a common memory location designated for a flag. If the flag is already set, then this processor must wait for the previous processor to finish with the resource. If the system is comprised of more than two processors, then, rather than polling the 'test and set' flag, the processor would leave its identity number (ID) in some predesignated shared memory ID area and then wait. The previous processor, after finishing with the resource, can use this ID number to inform the waiting processor that the resource is now available for its use. Alternatively, a central task awakening mechanism could be used to do this job. In this way the processors can be mutually excluded for common resource sharing and the waiting queues can be kept to a minimum. These waiting queues can be organised either on a first come first served basis or on some form of priority basis. In the case of the priority implementation, the processors can set their respective priority bits in the 'ID' area. The waiting processor with the highest priority could be immediately chosen and informed about the availability of the resource.

If a master/slave scheme is used and the resource is handled by the master only, then a software packet switching mechanism can be applied. The slave, instead of accessing the resource directly, can work out the data to be transferred, place it in a packet and leave the packet in the master's work queue. The master deals with the packets in the work queue on a turn by turn basis and later on, after finishing with each processor request, it sends the relevant packet information to the requesting processor.

The inter-processor interaction is also decided by the machine architecture. The

drawback of the shared memory type interaction mechanism is that the request from a processor may not get immediate attention. The initiating processor can set up the request flag for the target processor, but the target processor will only participate in the communication process or packet handling when it is scheduled for it. The initiating processor may keep on waiting for the target processor to be scheduled to receive the message or for an access to the resource. If the scheduling is done on the priority basis, the chances that a low priority processor will get an access would be even less.

In some machines, these interaction delays are reduced to a minimum by using an inter-processor interrupt mechanism. One faster way of using an inter-processor interrupt mechanism can be by implementing it in the hardware. For example, for a 32 processor system, a remote commonly accessible 33 bit interrupt register can be used as an ID area. One bit of this register can be allocated to each of the 32 processors as their priority coded ID bit. The last of the 33 bits of the register can be designated as 'flag' bit. This flag can be designated as 'if set the resource is not available'. The resource accessing processor, after finding its attempt as unsuccessful, sets its ID bit into the interrupt register. The processor holding the resource, after finishing the data transfers, resets its ID bit in the interrupt register and then clears the 'flag' bit. When this flag is cleared, an interrupt is automatically generated for the waiting processor with the highest priority in the interrupt register. The interrupted processor sets the flag and is, then, allowed to carry out the services with a minimum possible delay.

The inter-processor interrupt mechanism can be implemented efficiently in a machine with a limited number of processors. It is rather difficult to apply this technique in machines where a number of processing elements and other resource devices are connected to each other using two or more data buses. This is because

a different level of interrupt will be required to inform the processor which bus is available for its use. Also the limited number of interrupt levels on a processor will limit the number of buses which can be made available to that processor. For example, a processor which can handle four interrupts can not be informed about the availability of a fifth bus without adding extra hardware to it. Therefore, in loosely coupled machines with a single bus, or in some tightly coupled machines where the processors are hardware configured and are not allowed to be reconfigured (e.g. processor arrays), it is straight forward to hardwire these interrupts to the individual processors. In a multiple bus system, it will involve a predecided bus allocation topology which will become difficult to change later on without changing the hardware.

The aim of a parallel processor system is to provide a faster and more efficient service to the user. Different tasks or different parts of a task are managed in parallel and, if possible, execute concurrently. The system can work efficiently, if the work queues for the services can also be kept to a minimum. The inter-processor interrupt mechanism and a choice of the faster service devices can help to reduce the service time to some extent but it cannot cope with the higher speeds of present day microprocessors and data manipulation requirements. The logical solution would be to provide more buses within the system to ease the inter-processor interaction and also the addition of more facilities to provide parallel services. Providing more parallel buses within the system introduces complexity to the system. The INMOS transputer has several link to provide communication on the processor level. These links give a fast means of inter-processor communication on a two wire system with minimal wiring complexity. Fast parallel services can be provided by introducing parallelism at the 'service device' levels [29]. The next section gives a brief account in implementing the service device parallelism which was kept in mind during the design of the I/O

controller and the graphics systems.

2.2.3 The computer resources and their allocation

The resources in a computer can be classified as the hardware resources and the logical resources. The hardware resources include the physically available resources, i.e. the processor, the memory (both main and auxiliary) and different peripherals (e.g. printer etc). The logical resources are the abstract resources available to the user in the form of different utilities. These logical resources are shared differently among the different classes of users and are protected from unauthorised and inconsistent access. The operating system provides the methods for assigning these resources to the user and at the same time maintaining its availability to the other users. These other users may include other persons using the same computer in a multi-user computer network, or microprocessors in a multi-processor environment.

To manipulate the resource sharing effectively, the operating system needs adequate support from the available physical resources. The user may or may not be concerned about the nature or structure of the physical resources. For example, if the user program is long enough and can not reside in the main memory, it will require the support of the auxiliary memory device (i.e. hard disk) [30]. The user will not be concerned about which part of the disk unit is being used as long as the required data is available in the proper executable sequence. But in the case of a flag being set for some other user or task, it will need to make sure that the physical flag position is the same one that is known to both the initiator and the target, and also that the flag location is accessible to both. This is also the case if packet communication methods are used. The packet locations should be accessible by the initiator as well as the target.

If a task is using an auxiliary memory device along with the main memory because the main memory is not enough to accommodate the full length of code, the task will run slower. These delays are involved because of the loading and unloading of the code segments from one type of memory device to the other. These delays can be minimised by using specialised hardware for this purpose. A Memory Management Unit (MMU) along with a Direct Memory Access controller (DMA) can be used to transparently load required segment of code or unload unused segment of code on demand or concurrently with the running code. The delays involved because of the hard disk speed can be reduced by using more than one hard disk unit. This can be done by allocating the hard disk on a group basis. That is, for example, if 'n' processors comprise the parallel computer system and 'm' disk units are available for use, then a group of 'n/m' processors (where 'n/m' is an integer) can be allocated one hard disk unit (fig 2.8). The designed I/O controller (chapter 4, 5 and 6) can be organised to handle the unit for the individual groups. In this arrangement, however, the system will lack a central storage unit. This function can be fulfilled by either providing a separate central disk storage unit or by giving one of the 'm' disk drive units the status of the central storage device.

Another arrangement for a 32-bit bus system would be to use four disk drive units and directing the individual bytes to the individual drives (fig 2.9). With this Combined Disk Arrangement (CDA) four bytes of data can be retrieved from the disks in the time normally required for one byte. The 32-bit words can be directed to the four individual drives in parallel without involving any extra hardware. The same disk bank can also serve as central storage. The SCC68070, a Motorola 68000 compatible 32-bit processor with a 16-bit external bus, was used as an I/O controller for a transputer based computer. It was also provided with the facility of the disk controller. The on-chip DMA controller of

the SCC68070 can be organised for 8 bit or 16 bit transfer operations. Therefore the provisions can also be made to use two instead of four disk drive units in the above mentioned CDA arrangement with each of them storing two bytes. This way the CDA arrangement can be applied to use either four or two disk drive units. The DMA controllers can be programmed to carry out the required size data transfers.

The computer peripherals include the console, printer, mouse etc. These are all slow devices and do not interfere much with the processor speed. Another peripheral which is important and requires a higher service speed is the graphics terminal. The graphics information can be stored either in the bitmap pattern or it can be organised in a compressed file [31]. This information has to be processed before presentation to the output terminal. To obtain real-time picture effects, the processing speed requirements are higher compared with the data output speed requirements. The high data output speed can be achieved by the use of dedicated graphics controller chips [17]. The picture drawing algorithms obtain a high processing speed by using graphics controllers for low level graphics algorithms, e.g. line drawing, area filling etc. For animated moving pictures, a page flipping mechanism is widely used. In this arrangement the picture is updated on the page which is not being displayed. After updating the full page with the most recent information, it is flipped over and displayed while the page previously being displayed becomes available for the picture updating. To produce the real-time effects, these pages should be updated at least some 30 times in a second. This speed requires a higher data processing and data movement capability, which is usually restricted by the display memory bus bandwidth.

As mentioned earlier, the higher processing speed can be achieved by the parallel processing arrangements. The picture information can be distributed into differ-

ent sections and the information processing task is distributed among a number of processing elements. For example, one processor can be asked to draw a group of lines while the other is handling a different set of lines. These lines can be calculated in a separate memory and one processor can transfer the line information to the display memory by exclusively locking out the other processors while it is writing. The processor which has to draw the top most lines can be given a chance at the end. In this arrangement, because all the processors are using the same display memory bus bandwidth, they might find it hard to achieve the real-time effect. Again, if the graphics controller chip is also using the same bus bandwidth (i.e. a single ported RAM is used), this hardship could appear as a bottleneck. Because multi-port RAM devices are more expensive than single ported ones, it would be ideal to find a solution using a single port memory, provided the higher memory bandwidth can be achieved and the bottleneck can be avoided.

Another solution to achieve the higher picture processing speed is to distribute the picture screen into different sections. Each processing element can be allocated with one section of the picture to calculate. Sometimes it is not necessary to update every pixel on the screen, and part of the picture can be left as the static display. Therefore, depending upon the application, these picture sections can be made either of equal physical dimension, or they could be of equal logical size. That is, if one part of the picture is more processing extensive, it can be reduced in physical size to reduce the work load on the relevant processor. Similarly, the work load of the processor which is working on a relatively simple part of the screen can be increased by allocating it more physical area. This would make sure that all the processors have approximately same work load. This would give a higher processing speed compared to the situation when one processor, after finishing earlier, becomes idle while some of the others are still busy in calculating

their more dense areas. Again, in this arrangement, if the picture memory is a single ported one, the bus bandwidth limitations can become more obvious by increasing the number of the picture sections. The number of processors to which a picture can usefully be distributed is clearly dependent on the available memory bandwidth.

A solution to the above problem could be found if separate memory modules are used for the separate parts of the picture. The pixel output can be switched over from one memory module to the next at the end of the first section of the picture thus maintaining the consistency of the display. The processing elements can be given exclusive access to their separate memory modules which would also be part of the picture memory. This way the picture is divided physically and as the memory modules are hardwired, the logical distribution will be difficult. In this arrangement, however, the full memory bus bandwidth would be employed by the individual processor and could be exploited to the full extent.

The above approach was used to implement the Screen Level Parallel Graphics (SLPG) mechanism. The T800 was used as the picture processing element (chapter 7 and 8). The T800 is also given a separate 1.0 Mbytes of RAM and the operating system 'Helios' resided in this memory to maintain its compatibility with the rest of the computer. The VSC was used as the dedicated graphics controller and was provided with the 0.5 Mbytes of DRAM as the picture memory. The VSC also provided the DRAM refresh, the address multiplexing and the other signals (e.g. RASN, CASN etc) to control this DRAM. The system is made a modular one and for an 'n x m' picture, where 'n' is number of pixels per line and 'm' is number of lines per picture, it can be extended from one to 'm' processing elements in parallel by plugging in the extra boards (chapter 8). This way the parallelism in the SLPG mechanism was extendable up to the number

of the individual lines on the display.

2.2.4 Reliability and maintainability considerations

The reliability of a parallel processor system is, normally, measured as the fault tolerance, reconfiguration after failure, mean time between failures and the ability of graceful degradation. The mean time between failures in a system could be minimised by built in self-check mechanisms and diagnostic software. Faulty nodes can be diagnosed immediately and replaced. The modularity of the system makes repairs very easy.

The system can fail because of either hardware failures or software faults in the operating system. Hardware faults, although difficult to diagnose, are relatively easy to detect. For example, if a disk drive or a console is not working, the user would immediately know this. Similarly, if some part of the RAM or other hardware failed, the operating systems, normally, has the provision to report these faults immediately to the user. On the other hand, software faults are relatively difficult to recover from as they may cause an inconsistency in the data stored on the disk. If a machine is provided with a standby unit (e.g. complicated industrial controllers), the operating system's 'crash recovery' facility would be able to overcome the fault by switching on the standby unit. In a normal system where the hardware failure is not so critical, and because of the higher costs involved, these standby units are not provided. Instead, the operating system can be provided with some reconfigurability to make use of the healthy nodes by completely ignoring the faulty node.

The system reconfigurability, after a node failure, would depend upon the machine architecture. Considering the star configuration (fig 2.3), the work load on the

node in the centre is much more compared to the nodes in the corners. Failure of this central node, if the other processing nodes are not provided with some comparable capabilities, would mean a complete system failure. The machine would lack reconfigurability and therefore the system would not function at all with the failure of this single node. Similarly in case of the tree structure, the communication between different branches of the tree is handled by the processor at the top of these branches. A failure of the top processor could mean the failure of the full branch regardless of the fact that the nodes at the lower level of the branch are still healthy. This is because these processors are unable to maintain their communication with the other branches and to the I/O services. The emphasis is made on the reconfigurability of the I/O services because of the fact that the hard disk is providing the 'Disk Operating System' (DOS) information and other user utility files and routines. The I/O is also providing the user interaction in the form of console and graphics output.

A solution to the above problem could be to provide the hard disk and other I/O facilities to a number of processing nodes in parallel with the master node. The system could be provided with multiple buses to facilitate the inter-processor communication at the processor level to some extent, at the same time avoiding a full interconnection. This could introduce an element of fault tolerance within the system. Zaki and Elboraey [32], taking into account the basic MIMD interconnecting mechanisms, have concluded that the star and the tree structure become the least reliable with an increasing number of processors. The hyper-cubes (both dual bus and normal hyper-cubes) as well as the near neighbour mesh, maintain or even increase their reliability as more processors are added.

A good fault tolerant system is one which, in case of a node failure, is provided with sufficient reconfigurability to stay operational. In the event of a node failure,

the operating system, after diagnosing that a node has failed, can reallocate the work load of the faulty node to a healthy node. A built in supervisory task can help to do this job of dynamically allocating the task of a dead processor to an available healthy one. This would reduce the overall system speed, but would increase the reliability and would add an element of graceful degradation to the system. This would mean that the machine can withstand the node failures to a certain extent and that the failure of one or more nodes would not halt the system operation.

The hardware can be more reliable with the Very Large Scale Integrated circuit (VLSI) devices rather than the medium scale or large scale integrated circuits (MSI or LSI). This is because in case of VLSI, more of the circuit is confined within a chip and is less susceptible to the external noise sources because of the reduced number of the exposed conductors and the less complex printed circuit board. Also, the devices can be kept close to each other, which reduces the conductors length thus adding to the stability and hence the reliability. VLSI chips also include a larger part of the complete circuit than MSI and LSI devices. In case of a system failure, it is easier to diagnose which part of the circuit is malfunctioning rather than pinpointing the faulty registers and gates. An effort was made to build the I/O controller and the graphics board around available VLSI chips. In places where the individual gates could not be avoided, Programmable Array Logic (PAL) devices [33] and Programmable Logic Devices (PLDs) [34] were used to minimise the number of interconnections. Therefore, because of the lower number of chips involved, it would be easier to diagnose the faulty chip which in turn would reduce the repair time and help keep the system active for a longer time.

2.3 Summary

A brief survey of parallel processing requirements has been given. Parallel processing is a means of achieving higher processing speed. Both the multi-user, multi-processor and the single-user, multi-processor environments were taken into consideration. Only the Multiple Instruction, Multiple Data (MIMD) machines were considered because of the fact that these are becoming more popular compared with the Single Instruction Multiple Data (SIMD) machines. A brief account of the basic multi-processor architectures is given. The stability and the inter-processor communication for these architectures were also considered.

Two methods of manipulating a faster hard disk device have been suggested. One method would be to provide a hard disk to a group of processing nodes. This would would allow the implementation of a fast auxiliary memory technique. The second method would be the Combined Disk Arrangement (CDA). This would increase the overall data transfer rate to and from the hard disk. A method of introducing Screen Level Parallel Graphics (SLPG) was also given. It was shown that, with the designed hardware, the SLPG can be exploited up to level of the number of individual lines within the picture.

Chapter 3

Introduction to the SCC68070 and the VSC

3.1 Introduction

The SCC68070 [2] is a highly integrated 16 bit microprocessor. The CPU is based on the Motorola MC68000's internal structure, and is fully software compatible with it. Like the MC68000 [35] it has a 16 bit external and a 32 bit internal structure. It also has the same instruction set, programming model and internal resources as the MC68000. The clock cycles are also the same. The Interrupt Priority Level input signals (IPL0-IPL2) on the MC68000 are replaced on the SCC68070 with decoded external interrupt signals with separate acknowledge outputs. Two additional internally programmable interrupt pins are also available. These are served by the on-chip auto-vectors with an entry to a separate vector table, called the 'auto-vector table'.

The SCC68070 contains several on-chip peripherals. These peripherals include a Memory Management Unit (MMU), a two channel Direct Memory Access (DMA) controller, a serial interface, an 'inter-integrated circuit' (I2C) bus interface and a timer. The MMU can be used for protection against illegal access to the su-

pervisor state, as well as controlling read or write data transfers and instruction execution. This protection is provided on a segment basis. An illegal access causes a bus error on the CPU, and the MMU inhibits the data strobes to block the bus cycle.

The two DMA controller channels can perform data transfers in single address mode on channel 1, or either single or dual address modes on channel 2. The programming model is compatible with the existing Motorola 68000 family members (i.e. 68430, 68440 and 68450 DMA controllers). Both 8 bit or 16 bit data transfers are possible in either 'cycle steal' or 'burst' modes. The 16 bit wide transfer counter means that up to 65536 word blocks can be transferred within the full address range of 16 Mbytes.

The serial interface controller is provided on the SCC68070 with independent transmitter and receiver functions. Each function is programmable including the number of bits per character, number of 'stop' bits and baud rate. The inter-IC bus interface can be programmed for master or slave mode. It can address up to 128 different devices and can also be programmed to monitor bus activities. The timer provides one free running reference timer and two independent match, count and capture registers. These three modes (i.e. match, count and capture modes) can be programmed independently.

The SCN66470, Video and System Controller (VSC), provides an ideal counterpart for the SCC68070. It is designed to work with the 68000 family. It combines several functions to implement a basic computer along with the SCC68070 and the memory devices. It can address up to 2.0 Mbytes of memory including DRAM, ROM and memory mapped I/O peripherals. The DRAM refresh is also provided.

The VSC also includes an on-chip display controller which supports bit-map or compressed display files. It generates the required timing chain and synchronizing signals to produce video displays with a resolution of up to 768 x 560 pixels. The image can be changed by pixel manipulation logic, which can work under the control of either the CPU or a coprocessor.

The SCC68070 along with VSC and memory provides a small basic computer system with low system cost. This chapter gives a quick reference to the facilities available with the SCC68070 and the VSC.

3.2 The Central Processing Unit (CPU)

The programming model of the SCC68070 CPU is same as the MC68000. Therefore it is fully software compatible with MC68000. The bus error exception processing is, however, different from MC68000. In the case of a bus error, the SCC68070 stacks more internal information and, therefore, can provide full recovery. Also some instruction execution timing are different than MC68000. The features for the CPU can be summarised as follows.

MC68000 compatible bus interface.

Full MC68000 software compatible.

32 bit internal structure.

16 Mbytes addressing range.

14 addressing modes.

Memory mapped I/O.

Built in clock generator.

Programmable interrupt inputs.

Decoded interrupt acknowledge.

Vectored and auto-vectored interrupts.

The CPU has eight 32 bit data registers. These can be used for bit, byte, 16 bit word and 32 bit long word operands. The byte operands can be in either binary or BCD form. The lowest bit in each data register is referred to as bit 0. The data operations take place from the lower end of the register (i.e. byte operation is referred to the lower byte in the register), and for bit, byte and word operations the upper parts of the register remain unchanged.

Along with the data registers, the SCC68070 CPU possesses seven 32 bit general purpose address registers, two 32 bit stack pointer registers (the Supervisor Stack Pointer (SSP) and the User Stack Pointer (USP)), one 32 bit Program Counter (PC) and one 16 bit status register. The address registers do not support byte size operations. If the address register is the source operand, then 16 or 32 bits are used, but if it is a destination register all 32 bits are changed. Depending upon the processor state, either SSP or USP are accessible. The processor state (i.e. supervisor or user state) is reflected in the status register. The other bits in the status register include the condition code bits, the interrupt mask and the trace bit [2].

The SCC68070 can access the memory for byte, word and long word transfers. A byte can be at even or odd address boundaries, but words and long words are always aligned to the even boundaries. A word or long word transfer on an odd address will cause an address error exception. The program counter is

incremented automatically with the size of the operation (i.e. byte, word or long word).

3.2.1 The bus arbitration

The bus arbitration in the SCC68070 is similar to the MC68000. The Bus Request (BRN) and Bus Grant Acknowledge (BGACKN) signals are wire-ored. The Bus Grant (BGN) signals are daisy chained through the on-chip peripherals giving the on-chip DMA channel 1 the highest priority while the CPU is given the lowest priority. The BGN signal can also be daisy chained through the external devices according to their required priority levels.

When the CPU detects asserted BRN, it releases the bus after completing the 'present' bus cycle. BGN is asserted and the CPU tristates its local bus. The next bus master can respond to BGN by either asserting BGACKN or by not negating BRN. BGN is negated when both BRN and BGACKN are removed. If the CPU detects BRN negated before BGACKN, it negates the BGN and continues as normal. This arrangement reduces bus occupation problems because of noise.

After receiving the asserted BGN signal, the next bus master should wait until ASN, DTACKN and previous BGACKN are negated. It will then assert its own BGACKN. The devices are not allowed to 'break in' on the unfinished bus cycles. A bus master is considered a bus master until it negates its BGACKN. This, therefore, should be the last signal to be removed.

If BRN is still asserted after the BGACKN is negated, then the CPU issues another BGN and hence performs another bus arbitration for the next bus re-

questing device.

3.2.2 The exception processing

The CPU has two privilege states, the supervisor state and the user state. The two states choose between the two stack pointers (i.e. USP and SSP). The on-chip MMU can be programmed to translate the address between the two states and restrict the access to different code segments. The supervisor state is also used to access the on-chip peripherals. The 'S' bit in the status register is set for the supervisor state [2].

If the CPU is not in the supervisor state, then it is in the user state ('S' = 0). Some privilege instructions, such as 'stop', 'reset', 'move to and from USP' cannot be executed in the user state. Similarly, the status register cannot be over-written. Therefore only exception processing can change between these privilege states.

The two possible processing states are normal processing and exception processing. In the normal processing state the instructions are fetched and executed with the results stored in memory or the registers. If, however, an interrupt occurs or a 'TRAP' instruction is executed, the processor goes into the exception processing state. During exception processing, the 'current setting' of the 'S' bit is saved and the 'S' bit is asserted. This takes the processor into the supervisor state and allows execution of privileged instructions. The possible causes for an exception are bus error, reset, interrupt, address error, tracing, some instructions (e.g. TRAP, TRAPV) and privilege violations (i.e. trying to execute privilege instruction in the user state).

In the event of an interrupt, the interrupt priority mask is updated in the status register. After this, the exception vector number is determined. It is either found internally or acquired from the external logic. For example, when the interrupt is an acknowledged interrupt, the interrupt vector number is acquired from the interrupting device. If an auto-vectored interrupt occurs, the vector number is determined internally and the auto-vector table is used. TRAPS, bus errors etc all generate particular vector numbers internally. This vector number is then used to generate the address for the exception vector. The exception vector number is an eight bit number and is multiplied by four to give the exception vector location. The contents of these vector locations are the addresses of the routines which can handle the interrupt exception. The CPU starts exception processing by stacking the current processor state and program counter. It then enters the supervisor state. The Trace bit of the status register is disabled (i.e. 'T' = 0) so that the exception can run freely. The vector contents are fetched and loaded into the PC. Then execution of the exception is started at the new PC location.

The Return from Exception (RTE) instruction is used to end the exception processing. The reverse stack action takes place and normal processing is resumed. In the event of a bus error during stacking operation (in exception processing) the CPU is halted. Only the external 'reset' can end this halted state.

3.2.3 The interrupt structure

The MC68000 interrupt priority level signals (IPL0-2) have been replaced by decoded signals on the SCC68070. For this purpose the interrupt pins (IN2N, IN4N, IN5N and IN7N (i.e. Non Maskable Interrupt (NMI)) are available. These interrupt signals are individually acknowledged by the processor using the respec-

tive interrupt acknowledge pin (IACK2N, IACK4N, IACK5N or IACK7N). These interrupt signals are not latched and are required to stay asserted until acknowledged by the CPU. During the acknowledge cycle, the interrupting device can either provide the vector number on the data bus or can direct the processor to use the auto-vector table by asserting the Auto-Vector (AVN) signal. In case of an auto-vectored interrupt the processor generates the interrupt vector number internally according to the interrupt priority level.

Along with the decoded interrupt pins described above, the SCC68070 has two latched interrupt pins INT1N and INT2N. These have programmable priority levels which can be software programmed through the Latched Interrupt Register (LIR) [2]. These interrupt signals require no acknowledge cycle and are served through the on-chip auto-vector table.

The on-chip peripherals can be programmed to interrupt the CPU through the Peripheral Interrupt Control Registers PICR1 and PICR2. These registers can be programmed to interrupt the CPU on any required priority level. PICR1 is used for the Inter-IC and the timer interface while PICR2 is used for the serial interface [2]. The interrupt priority levels for the two DMA channels can be programmed in their respective Channel Control Registers [2].

The interrupts on the SCC68070 are serviced on priority basis. An interrupt with a priority level equal to or less than that being currently serviced is not accepted. 'NMIN' has the highest priority and is always serviced first. If the external and internal (on-chip) peripherals are programmed to the same priority levels, then a predefined daisy-chained priority is used. The interrupt service priority, in this case, will be NMIN (highest priority), INXN (IN5N, IN4N or IN2N), INT1N, INT2N, timer, serial interface (receiver and then transmitter), Inter-IC interface

and DMA (channel 1 and then channel 2).

3.2.4 The instruction set

The SCC68070 inherits the large and powerful instruction set of the MC68000. These instructions can support bit, byte (binarry or BCD), word and long word data operations in different addressing modes (Philips :68070). The length of instructions can vary from two bytes to four or six bytes depending upon the type of instruction and the address mode. The first word of instruction is the 'control' word or the 'operation' word. It indicates the type of instruction (e.g.MOVE, ADD etc), the size of operation (i.e. byte, word or long word) and where to find the data (e.g. register or external memory). The second and third words of the instruction (if applicable) contain the 'immediate operand', or the source or destination address.

Each instruction is required to be aligned on a word boundary. The two-word tightly coupled instruction fetch mechanism improves the performance. When the execution of an instruction begins, the operation word and the word following that have already been fetched. If the instruction can cause a branch, then the processor delays fetching the next instruction until after the branch decision has been taken. Similarly, in the event of an interrupt or any other exception (e.g. bus error), the fetched instruction in the pipe is not used.

3.3 On-chip peripherals

The SCC68070 has a separate bus and interface to the on-chip peripherals. This form a clean division between the CPU and the peripherals. These peripherals are independent of the CPU and work according to the 68000 standard bus protocol procedures. This on-chip integration of these devices has reduced the space and effort to implement these functions. These are memory mapped devices and are placed outside the 16 Mbytes address range (i.e. the outside world memory map) of the SCC68070.

The on-chip address bus consists of 32 bits. The upper two bits (A30 and A31) are used for the on-chip address decode. When the on-chip peripherals are decoded, the external bus lines are tristated and remain inactive.

3.3.1 Memory Management Unit (MMU)

The on-chip MMU of the SCC68070 is a subset of the MC68910 and MC68920 Memory Access Controllers (MAC) for the 68000 family. It can be used to translate logical addresses to physical addresses and for protection against unauthorized access to the memory. Protection can be provided to the individual segments depending on the conditions of supervisor, read, write or execute permission bits set in the MMU status register. Any unauthorized access to a segment results in a bus error condition generated by the MMU. The MMU can only influence the off-chip addresses from the CPU. The on-chip addresses and the addresses from the DMA are not affected.

The MMU can be enabled through the MMU control register [2]. Each memory

segment can be individually enabled or disabled by its descriptor in the on-chip 'descriptor RAM'. If the segment is disabled, it does not produce any match and it is treated as if the segment is not present. But, if the segment is enabled, it is tested for each possible access protection.

The MMU can operate in two different modes. In first mode, up to 8 segments can be defined with the individual maximum segment length of 2 Mbytes in 1 Kbyte blocks (i.e. a total of 2048 blocks per segment). In the second mode, the number of segments can be up to 128, but the segment size is reduced with a maximum of to 128 Kbytes (i.e. 128 blocks per segment). The segment descriptors are defined in the main memory and at any time a maximum of 8 segment descriptors can be down-loaded into the on-chip RAM. These segment descriptors must be first loaded and then enabled. Finally the MMU can be enabled.

The segment descriptor can also be loaded to the MMU RAM while the MMU is still enabled. In this case the descriptors must be disabled and should be validated only after they have been been loaded into the MMU RAM. This is to avoid any accidental access to a segment descriptor before it has been fully loaded.

3.3.2 The DMA controller

The SCC68070 also has two independent on-chip DMA controller channels [2]. They can be used to transfer data between a device and memory and can handle 8 bit or 16 bit data transfers. The channels can be programmed to operate in either cycle-steal or burst mode. The built in daisy chain gives the DMA controller a higher bus request priority than the CPU, with channel 1 at the highest priority.

Channel 1 can handle only single cycle data transfers while channel 2 can make either single cycle or dual cycle transfers.

The external device uses a DMA Request signal (REQ1N or REQ2N) to request data transfers. After receiving the request, the DMA controller acquires the bus and makes the data transfers using the Request/Acknowledge (REQnN/ACKnN) handshake signals.

The data transfers can take place either in single address or in dual address mode. In the single address mode, the data is transferred between memory and device in a single bus cycle. After acquiring the bus, the DMA controller waits for the Ready (RDYN) signal from the device and then puts a valid memory address on the address bus. For memory to device data transfers, valid data is available when data strobes (UDSN and LDSN) are asserted. For device to memory transfers, the data should be valid before REQnN and RDYN are both asserted, and should remain valid until the ACKnN signal is negated.

For the double address data transfer mode, an internal register is used. After acquiring the bus, the DMA controller puts the source address on the address bus. The CPU signals ASN, LDSN, UDSN, R/WN, and DTACKN are used as normal for the data transfer handshake. When the data on the data bus becomes valid (i.e. REQnN and RDYN or DTACKN are valid), it is stored in an internal DMA controller register. Then the DMA controller places the destination address on the address bus and the data is transferred to the destination with the required handshake.

The sixteen bit 'Memory Transfer Count' register (MTC) provides the data transfer count. This register is decremented with every transfer and the DMA con-

troller terminates the transfer when the count is zeroed. Then the 'Done' (DONEN) signal is asserted and the Channel Operation Complete (COC) bit in the channel status register is set. If the DMA controller is programmed to interrupt, an interrupt at the programmed priority level is generated. If a bus error occurs during the bus cycle, or the DMA controller operation is aborted through software, then the DMA controller sets the error and COC bit in the channel status register and terminates the data transfers. The type of error can be found by reading the channel error register.

3.3.3 Timer device

The timer device on the SCC68070 has three registers (TR0-TR2) [2]. TR0 acts as a 16 bit continuous reference timer. It is incremented at a rate of '16/Ckout' (where Ckout = crystal frequency/2). When the count reaches to a value of 'FFFFH', the overflow bit in the timer status register is set, the interrupt (if programmed in the PICR1 register) is generated and TR0 is loaded with the value in the reload register register. This was used as a continuous reference timer for 'Tripos' (the operating system loaded on the SCC68070).

In Tripos [36] the time is stored in 'days', 'mins' and 'ticks'. The 'mins' refers to the number of minutes starting from the last day's boundary. 'Ticks' are defined such that there are 3000 ticks per minute (50 ticks per second). A timer interrupt, therefore, is required every 20 ms to update the 'ticks' value. The reload register was loaded with the value 'F7DCH'. The PICR1 register was programmed to interrupt when TR0 is exhausted. Since the SCC68070 crystal frequency is 19.66 MHz, TR0 is incremented every 9.6 micro seconds (uS). The above reload value causes the TR0 register to overflow every 9.6 uS x 2083 pulses = 20ms.

The two other timer registers, TR1 and TR2, are identical 16 bit registers and can be individually programmed to one of the match, count or capture modes. The two registers function independently and are connected to the I/O pins 'T1' and 'T2' respectively. These registers (TR1 and TR2) can be enabled and programmed via the timer control register to monitor any transition on the respective pins.

The event counter mode can be used to count the occurrence of external 'events'. The 'event' is programmed by the event control bits in the timer control register. In the capture mode, on the occurrence of an event, the contents of TR0 are captured in TR1 or TR2. The match mode switches the I/O pins into output mode. An overflow in TR0 sets the output and when the value of TR0 becomes equal to the timer register (TR1 or TR2 which ever is in the match mode), the output is automatically reset. Thus pulses can be generated at the output pins (T1 or T2) with the required frequency and duty cycle.

After a reset the functions of TR1 and TR2 are inhibited. TR0 is set to zero and immediately starts counting. TR1 and TR2 should be programmed by the CPU before they can start functioning. This programming automatically enables TR1 and TR2. The timer status register can provide information about what event has occurred and on which channel. After reading the status, the timer status register should be reset to zero by the CPU as this is not automatic.

3.3.4 Inter-Integrated Circuit (I2C) bus

The Inter-IC (I2C) bus interface is provided on the SCC68070 to facilitate communication with other I2C devices [2]. The two pins Serial Clock (SCL) and

Serial Data (SDA) are dedicated to this purpose. The SCC68070's I2C interface can work in a number of modes. It can be either a bus master (transmitter or receiver), or it can be addressed as a slave device (again as a transmitter or receiver). If the interface is working as a transmitter, the CPU writes the data to be transmitted into the I2C data register. This data is then transmitted to the I2C serial bus with the MSB first. Similarly if it is in the receiver mode, the received serial data is clocked into the I2C data register, which when complete, can be read by the CPU.

The interface also has an eight bit address register. The seven bit slave address of the interface is stored in the seven MSBs of this register. The I2C status register reflects the most recent 'status' of the bus interface. The clock control register can be programmed to the appropriate clock rate for the data transmission (the maximum clock for the bus is 100 KHz).

The interface can be programmed to interrupt via PICR1. An interrupt is generated every time a complete byte is either transmitted or received. An interrupt is also generated if a general call address or the slave address is recognized, and also if the interface had started sending data but lost the bus. Whenever an interrupt occurs, the 'Peripheral Interrupt' (PIN) bit in the I2C status register is set to zero. The cause of the interrupt can be found in the I2C status register. The interface waits until the interrupt is serviced by the CPU. It jams the I2C bus by holding the SCL low and keeps it low until the CPU sets the PIN bit again to '1'.

The I2C interface can also be programmed as 'always selected'. This is achieved by setting 'bit 0' in the I2C address register. In this mode address comparison is inhibited and the interface will receive any byte transmitted on the bus. This

mode can be used to monitor the I2C bus activities for debugging purposes.

3.3.5 The serial interface

The SCC68070 is also equipped to provide an on-chip asynchronous serial interface [2] otherwise called a Universal Asynchronous Receiver Transmitter (UART). This interface can support both half and full duplex transmission. The interface functions are controlled through a set of programmable registers including the 'baud rate'. The transmitter and receiver can be programmed to have separate baud rates. The clock to generate this baud rate can be the SCC68070's system clock or some external source connected to the external clock pin XCK1. The SCC68070's system clock is divided by four before redividing it with the 'divisor factor', but if the external clock is used it is not prescaled by four. The maximum permissible frequency for this external clock is 5MHz.

The transmitter and receiver of the interface can be enabled or disabled individually using the UART command register. The data to be transmitted is first held in the transmit hold register. It is then framed with the programmed 'start', 'stop' and (if enabled) 'parity' bits before being transmitted. Similarly, the received data is first held in the receive hold register. If enabled, a parity check is made on the received data. An interrupt (if programmed in the PICR2 register) is generated at the programmed priority level and the results are present in the UART status register for inspection.

3.4 Video and System Controller (VSC)

The SCN66470 Video and System Controller (VSC) [17] is a VLSI device which is designed to work together with the 68000 family to implement basic computer functions. It can be used with the SCC68070 to work as the system logic controller, DRAM controller (including address multiplexing and DRAM refreshing), display controller and coprocessor interface controller. The VSC is connected to the SCC68070 via a 20 bit address bus and a 16 bit data bus. This gives the VSC a controlled address range of 2 Mbytes. This address rage is split into 1.5 Mbytes of RAM and 0.5 Mbytes of ROM, internal registers and I/O area. The SCC68070 control signals (UDSN, LDSN, R/WN, DTACKN, BERRN, HALTN and ASN) are used for handshake purposes. The VSC memory map is given in Appendix A and the VSC registers are given in Appendix B.

The three DRAM banks (bank 1, bank 2 and bank 3) are 0.5 Mbytes each and are separated by three column address strobes (CAS0N, CAS1N and CAS2N). The VSC generates the DTACKN signal for the DRAM, ROM, DRAMIO devices and the internal registers. The generation of DTACKN is programmed in the 'Command and Status Register' (CSR), and can be delayed according to the speed requirements of the memory system. Normally this signal is generated when the data is valid on the system data bus. But if an 'Early DTACKN' is enabled (ED = 1 in CSR), then the DTACKN signal is generated two time slots before the normal DTACKN. For 'Early Write' (EW = 1 in CSR), DTACKN can be further advanced by two time slots.

The DTACKN for ROM is delayed by programming the bits DD, DD1 and DD2 in the CSR register (Appendix C). Whenever the ROM is decoded, the CSROMN

signal is also asserted. After reset, the ROM is swapped with the DRAM for the first four CPU accesses. This allows the CPU to fetch the initial stack pointer and program counter values from the ROM. These values, therefore, must be placed as the first eight bytes in the ROM.

The I/O devices are connected on the system bus. The VSC generates the 'Chip Select I/O' (CSION) decoded signal which can be used for the address decode of devices using the VSC controlled I/O address space. The VSC does not acknowledge accesses to this I/O space. Instead the connected devices are expected to generate acknowledge signals themselves.

The CSION signal is also generated for the designated DRAMIO address space. The DRAMIO area is used for devices connected to the DRAM bus. For these devices the VSC uses the memory address and data buses in the same way as for the DRAM access. The VSC generates an early or late DTACKN similar to that generated for the DRAM devices.

Another service provided by the VSC is a watch dog timer for bus hang up. When enabled (BE = 1 in CSR), the watch dog timer becomes active on the assertion of UDSN or LDSN. If the strobes remain low and are not acknowledged during one full video line (i.e. 64us), then a bus error is generated. The BERRN pin is driven low and 'BE = 1' is asserted in the CSR status register (Appendix C). Reading the CSR automatically resets the BE bit. The watch dog timer works from the data strobes, therefore, it remains active even if the addressed resource was not VSC mapped.

The VSC also generates the 'RSTOUTN' and 'HALTN' signals for the CPU. The 'Reset Input pin' (RSTINN) on the VSC, when pulled low, generates the

reset condition for the CPU. Both the 'RSTOUTN' and 'HALTN' signals are driven low and held low for 8 video frame periods (i.e. 8 x 20 ms = 160 ms). A capacitor connected to 'RSTINN' holds it low for an initial power-on period. This initializes the power-on reset procedure. A simple 'single pole single throw' (SPST) switch was added to the RSTINN pin as an optional manual reset (fig 3.1).

The VSC incorporates an on-chip display controller. Pictures are generated for several predefined programmable modes. The 'Video Start Register' (VSR) points to the display buffer within the display memory (i.e. DRAM controlled by the VSC). The pictures generated can be in interlaced or non-interlaced mode, with a programmable image frequency of 50 or 60Hz. The picture size can also be reduced to put some colour around the boundary of the picture using the border colour register. Two image control areas, 'Image Control Area' (ICA) and 'Dynamic Control Area' (DCA), exist in the memory. These can be used for dynamic picture manipulation. The Display Control Registers (DCRs) can be reprogrammed at the end of each frame through the ICA, and at the end of each line through the DCA (Appendix B). This programmability was exploited to produce the 'Screen Level Parallel Graphics'.

The VSC display controller generates the pixel output along with the composite video synchronizing signal 'CSYNCN'. The display files can be stored as bitmaps or compressed files (i.e. run-length or MOSAIC files). The compressed way of storing files is popular because it requires less space to store the picture information .

A pixel accelerator is also included in the VSC to speed up pixel manipulation. This can be used by either the CPU or a coprocessor. It can be used for testing

and modifying pixel contents, masking the destination pixel, barrel shifting and many logical operations such as swap, copy and patch, and compare.

An external coprocessor can be interfaced with the VSC via the memory data bus and the control lines. The Cycle Request (CYREQN) and Cycle Acknowledge (CYACKN) signals are used to synchronise the information transfer. The coprocessor accesses are provided within the CPU access window.

3.5 The microcore board

The microcore board is an evaluation circuit from Philips Components Ltd [37]. The components include a SCC68070 running at 9.83 MHz (19.6608 MHz crystal), SCN66470 VSC (controlling the DRAM, ROM and I/O resources), 0.5 Mbyte of DRAM and 128 Kbytes of ROM (including character font memory). An RS 232-C level translator is also included for console interface to the SCC68070's UART. The block diagram of the microcore is given in figure 3.2 and the circuit diagram is given in figure 3.3. The system bus and DRAM bus signals are available on the 96-pin 'euro-connector' (fig 3.4).

The microcore ROM was replaced by the Tripos bootstrap ROM. The RAM was extended to the full mega byte (fig 3.5). The VSC was setup to display pictures in 8 bits per pixel mode. A switch (S2) was added for manual reset and another switch (S1) on NMI was added to generate console interrupts to the CPU for debug purposes (Fig 3.1).

This board provided a 'ready built' minimum chip count computer system which included the SCC68070 and the VSC. This board was used as a base for further

developments.

3.6 Summary

An overview of the SCC68070, the main I/O processor used in the project has been given. The on-chip peripherals are reviewed as a reference. The internal CPU structure of the SCC68070 is similar to the MC68000, but the interrupt structure is different. The programming model is the same as the MC68000, therefore, software developed for the MC68000 can be used directly for the SCC68070. The peripherals on the SCC68070 are mostly subsets of their counterparts in the MC68000 family. This integration of peripherals along with the CPU on to one chip has reduced the 'foot print' and also minimized the electrical noise problems inherent in the printed circuit board.

Another VLSI chip, the SCC66470, has also been surveyed. Like the SCC68070, it is a highly integrated device which combines many functions for the implementation of a compact yet powerful computer system. The VSC includes the video controller which can work with bitmap as well as compressed files.

In an effort to built a computer using the above mentioned devices it was necessary to add both floppy disk and hard disk. The next two chapters discuss these two pieces of circuitry respectively.

Chapter 4

Floppy disk controller interface

4.1 Introduction

A computer must have a storage device, so that the information can be stored in it and later retrieved when required. The two types of storage media in general use are the tape device and the disk device. The tape, being a sequential device, is slower than the disk. The disk is a random access device and operates much faster than the tape. The computers, therefore, mostly use disk for on-line storage and tape for 'backup' or off-line storage.

Many disk storage media are in use today, ranging from a large selection of hard disks, floppy disks and also optical disks. With the exception of optical disks, all of these systems rely on the basic principle of magnetic storage. The disk surface is coated with a ferro-magnetic layer and a read/write head is used to pass information to and from this magnetic material. In the case of a hard disk, a rigid non-flexible disk is used. A read/write head flying just above the disk surface can extract and store information at a speed of the order of tens of Mbits per second. The whole assembly is enclosed in a box to make it safer and prevent damage from dust and other contaminants.

In the case of the floppy disk a flexible mylar disk is used. For ease of use this is permanently enclosed in a plastic jacket. A slit is made in this jacket to let the read/write head make contact with the disk. The plastic jacket prevents dust and grease from reaching the disk surface which can cause damage.

The data on the disk is stored in the form of a series of pulses called 'write data' pulses. When this data is read, it is also in the same form. An interface circuit is always required to control the head movements of the drive unit and to convert the 'raw' data into more a recognisable form for the computer. This is called a Floppy Disk Controller (or FDC interface).

This chapter gives the details of three alternative circuits that were investigated as options to control the floppy disk drives. The first circuit was built around the SCN68454 [38] . This is an intelligent device and can handle more than one type of disk drives (i.e. hard disk drives as well as floppy disk drives). The second circuit was built using one member of the WD279X family, the WD2793 [39]. As all these family members have roughly the same characteristics, the circuit was designed to accommodate any of the family members. The third circuit used the DP8474 [40] the stand alone floppy disk controller. The on-chip hardware and software control functions make it cheaper to implement and easier to use. The design of this circuit is also discussed. The comparison between the three FDC interface circuits is given at the end of the chapter. The circuit using the DP8474 was chosen as the ultimate FDC interface for the I/O board. The reason for this choice is also given.

4.2 Interfacing the SCN68454 (IMDC)

The SCN68454 is produced by Signetics using MOS-VLSI technology. It offers quite attractive features. Due to its enhanced capabilities, it is also named as the intelligent Multiple Disk Controller (IMDC). Some of these features are listed below.

bus compatible with 68000 family,

automatic re-run on bus error,

31 bit address counter,

8 bit or 16 bit selectable data transfers,

can support up to four hard disks or floppy disks or both in any combination,

can directly support SA-1000 and ST-506 hard disk interfaces,

can support both FM and MFM data storing methods,

data rates of up to 10 Mbits/sec for MFM and up to 2 Mbits/sec for FM are possible,

on-chip 128 byte FIFO buffer and on-chip DMA controller for fast data transfers,

memory accessible range of up to 4 Mbytes for DMA transfers,

multiple sector write and read with implied seek.

4.2.1 The IMDC register map

The IMDC has seven 8 bit registers. These registers appear as normal memory locations to the SCC68070 (CPU). The CPU can access the registers whether IMDC is in a 'busy' state or in an 'idle' state. The contents of these registers are read or written on the data bus (D0-D15). These are 'Event Control Area Pointer' (ECAP) register (four 8 bit registers organised as ECAP High, ECAP Middle High, ECAP Middle Low and ECAP Low), Interrupt Source Register (ISR), Interrupt Vector Register (IVR) and the Device Status and Control Register (DSCR).

The ECAP register by the IMDC as a pointer to a table of four long words in the system memory. These four long words are in fact pointers to four different Event Control Area blocks (ECA blocks). These blocks are arranged in ascending order by the drive numbers. The first long word points to the ECA block for 'drive 0' while the fourth points to the ECA block for 'drive 3' (fig 4.3). This is a read/write register and is cleared (to zero) after 'reset' is asserted.

The IVR register can be used to store an 8 bit interrupt vector number. On interrupt acknowledge from the CPU, the IMDC responds by putting this value onto the data bus. This facilitates vectored interrupts. This facility was not used in the work described here. Instead the CPU's on-chip auto-vector was used. The interrupt on the IMDC is cleared by reading the ISR register. The acknowledge cycle, therefore, was not required and hence no value was placed in this register. It is a read/ write register and a 'reset' sets it to '0FH'.

The interrupt source register provides the information about which drive has caused a previous interrupt. Bit 4 of this register is set if the interrupt was from

drive 0 and bit 7 is set if it was from drive 3 (fig 4.1). This is also a read/write register and the corresponding drive bit should be cleared by the CPU before issuing any new command. Reading this register after an interrupt clears the interrupt but does not reset the corresponding drive bits in this register. On reset the contents are automatically cleared to 'zero'.

Bit 0 in the DSCR register defines the transferring data width. Both 8 bit (bit 0 = 0) and 16 bit (bit 0 = 1) data transfers are possible. In 8 bit mode, only the lower half of the data bus is used. The upper nibble of this register (bit 4 to bit 7) is used to initiate the command operations for the drives (fig 4.1). When the respective bit for the drive in the DSCR register is set to '1', the IMDC starts the requested command operation for that drive. A new command can not be issued unless the previous command is completed and the corresponding drive bit in both the ISR and DSCR registers has been cleared in the same order. The CPU can, however, abort a command by using a 'force reset' of the drive bit in the DSCR. Again, the register is a read/write register and becomes 'zero' after reset.

4.2.2 The IMDC command procedure

The SCN68454 is designed as a memory transfer oriented device. The CPU transfers minimal information to the IMDC (a maximum of 7 bytes) before 'activating' it. Data read from the disk drive is put directly into the system memory by using the IMDC on-chip DMA controller. Similarly, a write to disk is performed directly from the memory. The ECA block, also in the system memory, is accessed and the information is acquired to control the data transfers.

The command procedure for the IMDC can be subdivided into three phases. These are 'initiating the command', the 'command execution' and the 'end of command'. Before initiating a command, the disk controller is either in an 'idle' state (i.e. no command is in progress) or it is already busy executing a command. In both cases the command can be initiated. If the disk controller is in the idle state, it will immediately start executing this 'new' command. If the controller was already executing another command, it will finish with the previous command before starting the new command.

To initiate a command on the IMDC, the CPU first sets up an 'ECA block' in the system memory (fig 4.2). The structure of this block is similar for all commands (i.e. read, write format command), but the information placed inside is different. After setting up the 'ECA block', the CPU sets up an ECA pointer's table in the memory. This table consists of four long words, each word pointing to an 'ECA block' for an individual drive (fig 4.3). After this, the CPU would set up the appropriate '8 bit' or '16 bit' data transfer mode. This is done by setting bit 0 in the DSCR register for 16 bit, and clearing bit 0 for 8 bit data transfers. After this (if applicable) the interrupt vector value would be placed in the IVR register and the ECA pointer's table address is loaded into the ECAP register. The CPU sets the 'busy' bit in the DSCR register for the appropriate drive, which in turn would initiate the command procedure. The IMDC starts performing the requested command on the particular drive.

Before setting the busy bit in DSCR, the drive should be in the 'ready' condition. If the command is initiated while the drive is not ready, the command is aborted and the 'End Of Command' (EOC) procedure is started. Similarly, if the command is in progress and the drive goes from 'ready' to a 'non-ready' state, the IMDC will abort the command and start the EOC procedure. If the

IMDC is already busy with a previous command and a new command is issued for a different 'idle' drive, the IMDC will finish with the previous command and then, after the interrupt has been cleared, immediately start with the command for the next drive. The new command can be initiated only for an 'idle' drive. The DSCR and IVR registers provide information in this respect. If the next command is for the same drive on which the previous command is in progress, the CPU will wait until the first command is completed. After the command has finished, the CPU will reset the 'drive bits' in both ISR and DSCR registers, and then set the drive bit again in DSCR to execute the next command. The pending commands are executed in ascending drive number order of priority. Therefore, the 'drive 0' is given the highest priority while 'drive 4' has the lowest priority.

When command execution begins, the IMDC reads the ECA pointers' table to locate the appropriate ECA block for the required drive. The information inside the ECA block is used to configure the drive parameters and command parameters. Once the necessary information is gathered (from the ECA block), the IMDC goes into 'local' mode. During this mode it configures the disk drive and tries to execute the command. If it was a 'read' command, then the data is read into the on-chip 'First In First Out' (FIFO) buffer before being transferred to the system memory using the DMA mode. If it is a 'write' command, the data is read from the system memory into the FIFO buffer using the DMA mode. This data is then written onto the disk surface in the form of 'write data' pulses during the 'local' mode.

For the DMA mode, the IMDC acquires the 'system bus' using '68000 standard' bus arbitration protocol (fig 4.4). The 'Bus Request' (BRN) is asserted and the IMDC waits for the 'Bus Grant' (BGN) signal. When the bus becomes free, it acknowledges the bus grant (i.e. it asserts the Bus Grant ACKnowledge

'BGACKN' signal) and starts the data transfer to or from memory. After the 'command' is completed, the IMDC updates the ECA block. This updated ECA block can be used to decide the status of the previous command. In the case of an error, the 'number of tries' byte in the ECA block is decremented on each try and is consumed before the IMDC sets any error message in the ECA block. An interrupt is issued at this stage to indicate that the data transfer phase is over. This also marks the beginning of EOC procedure.

During the EOC phase, the interrupt is cleared when CPU reads the DSCR. No 'interrupt acknowledge' cycle is required, but if the host initiates an interrupt acknowledge cycle, the IMDC puts the contents of IVR register on the data bus and removes the interrupt. The drive bits in both ISR and the DSCR register should be cleared before the IMDC is ready to deal with the pending requests or to receive any further commands. This EOC occurs both when the command was successful or it has failed.

4.2.3 Hardware considerations

From the hardware point of view, the IMDC operates in four different modes. These modes are named according to the type of data available on the 16 bit data bus. Same bus is used to supply addresses and data for the memory interface as well as reading and writing data to and from the disk device. These are the 'regular', 'local', 'DMA' and the 'idle' modes.

In 'regular' mode, the host can access the IMDC registers. A valid address is placed on the address lines A1 and A2, and the two data strobes (Upper Data Strobe (UDSN) and the Lower Data Strobe (LDSN)) are asserted. The Address

Strobe (ASN) is then asserted to indicate to the IMDC that the address is valid. Finally the 'Chip Select' (CSN) is asserted. Data transfers take place according to the state of the 'Read/Write' (R/WN) signal (a high R/WN for a read and the low R/WN for write). The bus size (i.e. 8 bit or 16 bit) is determined by bit 0 in the DSCR register. The IMDC generates the 'Data Acknowledge' (DTACKN) signal to end the bus cycle. This is a relatively simple mode and requires no external hardware other than the decoding of the CSN signal.

During the local mode the data is transferred between the IMDC and the disk unit. The same data bus (D0-D15) is used as the input as well as the output port. 'Enable 0' and 'Enable 1' (EN0 and EN1) signals are used to inform the external hardware about the state of the bus configuration. The data bus definitions as input and output port are given in figure 4.12. The output port signals are latched to assure their availability to the disk drive during the command execution. Similarly, external buffers are required at the input port so that the IMDC is able to configure and read the input port data whenever required.

The IMDC uses the DMA mode for data transfers to and from memory. It acquires the bus using previously mentioned protocol and asserts the 'OWNN' signal. This signal is used to control the direction of the 'control bus' buffers. For data transfers, the IMDC would first place the two upper address bytes (A19-A31) on the data bus and assert the 'Upper Address Strobe' (UASN). These address bytes should be latched in some external latches synchronised with the UASN. If the upper address range is not required then this step is not executed. Next, the lower address bytes (A3-A18) will appear on the data bus and the IMDC will assert the Lower Address Strobe (LASN) signal. Again the two address should be latched in some external latch using LASN. Along with A3- A18, A1 and A2 are also placed on the address bus. The R/WN and ASN signals are asserted

to validate the address. Data direction is determined by the R/WN signal. (A high R/WN signal means a read from memory and vice versa). The data size is determined by the two data strobes (UDSN and LDSN) which in turn are controlled by 'bit 0' in the DSCR register. A DTACKN from the memory device ends the bus cycle.

The IMDC starts the bus cycle termination by negating the ASN, UDSN and LDSN signals. DTACKN is expected to stay asserted until after the negation of the ASN signal. The above process would complete one DMA data transfer cycle. The number of DMA transfers for one bus acquisition cycle depends on the 'DMA transfer' byte, set inside the ECA block (fig 4.2). A maximum of 16 memory transfers are allowed at a time. After completing these data transfers, the IMDC releases the bus by negating the BGACKN signal. A fresh BRN signal will be asserted for the next set of data transfers.

The DMA phase is executed with minimum disturbance to the CPU. The IMDC's internal processor operates at a clock period three times the external clock input period. (i.e. Tproc = 3 x Tclock). Therefore, with an IMDC operating at a 16 MHz clock frequency (i.e. Tclock = 62.5 nS), the IMDC processor clock period would be (62.5 x 3 =) 0.19 micro second (uS). A single DMA transfer takes a maximum of three IMDC processor clock cycles (one for each, putting the upper address and the lower address words onto the data bus (D0-D15), and one for the data transfer). Therefore, a single DMA transfer will take about (0.19 uS x 3 =) 0.57 uS provided the memory device does not slow down the data transfer process. Supposing a floppy disk drive with a data transfer rate of 250 KBits/sec, a '16 bits' word would be ready in about 64.0 uS. Since a maximum of sixteen DMA transfers can take place at any one time, the time required to gather 16 words from the disk drive will be (16 x 64.0 uS =) 1024 uS. The IMDC

can make these 16 transfers (if 16 bit data size is selected) in (16 x 0.57 uS =) 9.12 uS. Therefore, the maximum time the IMDC can do without the system bus in between the transfers is (1024 uS - 9.12 uS =) 1014.88 uS. Violation of these timings can cause data 'overrun' or 'underrun' errors. These overrun and underrun errors can be avoided by assigning the IMDC a higher bus priority. The IMDC, however, will be able to stay up to a maximum of (128 bytes FIFO x 32 uS per byte from the disk) 4096 uS without the system bus during the entire single 'read' operation. After this time the on-chip FIFO buffer will be full and data 'overrun' error will occur. During the 'write' operation the IMDC needs a continuous data inflow, one byte every 32 uS or less. As the disk drive requires a data byte to be written onto the disk surface within the above mentioned time, no delay is acceptable.

The fourth mode is the 'idle' mode. Obviously the IMDC is not doing anything during this mode. Most of the control lines enter their inactive state. This mode is automatically finished by the next regular mode.

Based upon the four operating modes, a circuit was designed to use the IMDC as a Floppy Disk Controller (FDC) interfaced to the SCC68070. The circuit consisted of a number of latches and buffers along with the IMDC, DPLL and a flip-flop arrangement (to replace the Voltage Controlled Oscillator 'VCO') (fig 4.5). The two flip-flops and a counter were used to synchronise the read data pulses from the disk unit. A VCO was not included in the circuit and, therefore, is not included in the discussion.

4.2.4 Disk Phase Lock Loop (DPLL)

The DPLL, SCB68459 [41], is a companion chip to be used alongside the IMDC. It locks onto the incoming 'read data' pulses from the disk and, with the help of the 'VCO clock', it generates the composite 'Read Data' (REDAT) and 'Read Clock' (RCLOCK) signals for the IMDC (fig 4.6). Similarly, it takes the 'Write Clock' (WCLOCK) and 'Write Data' (WRDAT) signals from the IMDC and generates the 'write data' pulses which are written onto the disk surface (fig 4.7). It also uses the 'late' signal from the off-chip delay circuit to provide the write precompensation. If all four of the interfaced disk units have a same data rate, then only one DPLL chip is required. Otherwise one DPLL is needed for each type of disk drive. In this case a separate 'Decoder/Selector' circuit is required to choose between the DPLLs (fig 4.8).

The 'flip-flop and counter' arrangement, which was used to replace the VCO (fig 4.5), should provide the DPLL with a clock frequency which is twice the disk drive data rate. In this case, the floppy disk drive had a data rate of 250 Kbits/second. A clock frequency of 500 KHz was, therefore, required for the DPLL operations. The VCO clock input for the DPLL is also required to be 500 KHz. In the case of a VCO attachment, pump-up, pump-down and frequency-pump signals are used by the DPLL to 'trim' the VCO output to the required frequency. In this case these signals were not used. An arrangement was made to provide the required 500 KHz by the above mentioned digital method.

In the above mentioned arrangement, a 4 bit binary counter was synchronised with the 'read data' pulses from the floppy disk to provide the required frequency input signal for the DPLL. This 'digital' solution was only possible for the floppy disk controller interface as the floppy disk drives operate much slower compared to

hard disks. The two flip-flops and four bit binary counter generate the reference clock. When using the VCO, the read pulses from the disk unit are conditioned by the DPLL. The DPLL then generates the error signal (for the VCO) to the required phase relationship. The relationship between the phases is such that the rising edge from the read data pulses should occur with the falling edge of the VCO clock (fig 4.9). The DPLL then generates the REDAT and RCLOCK signals for the IMDC synchronised with the read data and VCO clock.

In this circuit, the first flip-flop clocks '1' (logic '1') on the rising edge of the 'read data' signal. This presents a '1' to the second flip-flop, which is clocked at 32 times the data rate or 16 times the DPLL clock rate (fig 4.10). Working within the error margin (i.e. approximately 3% of the data rate), the second flip-flop generates the 'load' signal for the counter and also clears the first flip-flop. The next clock edge loads a preset (binary) value of '0001' into the counter and also clears the second flip-flop (fig 4.10). The counter, configured as a free running 'divide by 16 counter', generates a square wave of 500 KHz at the 'Qd' output. This 500 KHz 50% duty cycle signal replaces the 'VCO clock' input required for the DPLL. When a read data pulse is received, it forces (through flip-flops 1 and 2) the 'Qd' output of the counter to zero thus providing the required falling edge. The counter starts counting and at (binary) '1000', the Qd becomes asserted, thus giving the required square wave output. Once synchronised, the counter should remain synchronised. In case of a 'jitter', the flip-flops will synchronise the system again.

The 'flip-flop stage' used as a synchronising circuit, introduce an error between 0.5 and 1.5 of the input clock periods for the counter. The average of these (i.e. 'one') was loaded into the counter in the beginning to give the best possible correction.

4.2.5 Circuit description

The complete circuit diagram of the IMDC implemented as a floppy disk controller is given in figure 4.11. To reset the system, the reset line is pulled low. This resets the IMDC along with the rest of the system. When this signal is removed, the IMDC executes an internal 'microprogram' to initialise itself. It latches '00H' in the output port latch 'C1', and the internal registers are initialised to their reset values. The IMDC then goes into the idle mode and stays 'dormant' until a command sequence is initiated.

In the idle state CSN is inactive (i.e. logic high). Similarly, the 'OWNN' signal is high and the 'Local' signal is low. The data direction in the transceiver 'B1' is controlled by the OWNN signal (fig 4.11). At this stage the buffer is enabled with the data direction towards the IMDC. The output of the address latches 'LA1, LA2 and LA3' are also disabled because of the OWNN signal. The local signal is used to enable the data buffers 'BD1 and BD2'. These are enabled in the idle mode, but the 'data direction' is towards the IMDC. Therefore, they present no problem to the system data bus and the host performs normally.

To make any disk access, the CPU initiates an IMDC command procedure. First, using the regular mode, the CPU accesses the IMDC registers to read the drive status and to write the required information into these registers. This is performed normally with no signals asserted by the IMDC other than 'Data Direction' (DDIR) and DTACKN. The DDIR signal from the IMDC is generated according to the state of the R/WN signal and is used to control the data flow direction in the data buffers BD1 and BD2. Other IMDC controlled signals, such as OWNN, Local,UASN and LASN remain in the same state.

After the command initiation process is over, the IMDC goes into the DMA mode. It asserts the BRN signal and after becoming the bus master, it puts the higher address byte (if applicable) onto the data bus and asserts the UASN signal. This is used to latch the upper address byte in the address latch LA3. Only one latch '74LS373' was used in this address range and only two address lines A19 and A20 were latched to give the IMDC access to the full 1.0 Mbytes of available memory. After the upper address, the IMDC puts the two lower address bytes (A3-A18) on the data bus and asserts LASN. This LASN is used to latch two lower address bytes in the LA1 and LA2 latches. Meanwhile, the IMDC places the two address lines 'A1' and 'A2' on the address bus and asserts UDSN, LDSN, R/WN and ASN according to the data transfer requirements. The first information that the IMDC reads from system memory is the address for the required 'ECA block' from the 'ECA pointers' table'. The memory access process is repeated until the IMDC has acquired all the information it requires to configure the disk drive. The IMDC then enters the local mode and starts the disk drive access.

During the local mode, the address latches (LA1-LA3) are again disabled and the data buffers (BD1-BD2) are open with the data direction towards the IMDC. This action leaves the bus available for CPU activities while the IMDC is configuring the disk drives. The address lines A3 through A8 along with the 'Chip Select for I/O' (CSION) signal were decoded to put the IMDC registers at the following addresses.

ECAP register = 1FFC00 32 bit,

IVR register = 1FFC04,

ISR register = 1FFC05,

DSCR register = 1FFC06.

The previously described ICs complete the IMDC interface with the system bus (i.e. to the CPU and memory). On the disk interface side, 'LC1' was used to latch the 'drive control' information. This information has to remain valid on the disk drive bus throughout the data transfer between disk drive and the IMDC. Since these signals are positive logic on the IMDC (i.e. active high), the inverters were used (LC3) to make them compatible with the disk drive. This also improved the current drive capability of the interface circuit. A buffer IC (LC2) was used for the input port. When enabled (by 'EN1') it provides the IMDC with all the signals required to know the status at the disk drive end. The buffer LC2 also invertes the input signals to make them compatible with the IMDC. One set of these inverter buffers is enabled permanently (pin 19 to ground) to provide the signals which are always required by both the DPLL and the IMDC (e.g. Index, READ DATA signals). These are separate inputs signals to both the IMDC and the DPLL and are not required on the configured input port.

The 74S124 along with a 16 MHz crystal provides a 16 MHz 50% duty cycle clock for the IMDC internal operations. The same clock was further subdivided by a 74LS393 (dual binary counter) to produce an 8 MHz signal used with the 'flip-flop and counter' arrangement to replace the VCO. The same 74LS393 also provides a 500 KHz clock required by the DPLL integrated circuit.

The write precompensation was provided by the two off-chip delay lines DEL1 and DEL2. The manufacturer's recommended precompensation (of 180 nS) was provided. This precompensation is enabled by the IMDC at its output port (fig 4.12) at a precompensation boundary track as mentioned in the 'ECA block' controlling the disk drive.

4.3 Interfacing the WD279X

As the name implies, WD279X family of disk controllers' [39] comes from the 'Western Digital Corporation' (a VLSI manufacturing company). This family includes the WD2791, WD2793, WD2795 and WD2797 floppy disk controllers. These family members differ in their data presentation on the data bus. The WD2791 and WD2795 have an inverted data bus while the WD2793 and WD2797 have a true data bus. Also the WD2795 and WD2797 have a side (i.e. head) select output. The common features of the family are summarised below.

on-chip Phase Lock Loop (PLL) data separator,

on-chip write precompensation logic,

single +5 volt power supply requirement,

TTL compatible input and output,

programmable automatic seek, and a verify after seek,

multiple sector read and write capability,

can interface 8 inch or 5.25 inch soft sectored floppy disk drives,

programmable control on 'head load' timing.

These devices have a number of registers including a status register, command register, track register, sector register and data register. The CPU is interfaced through an 8 bit bi-directional data bus. The data transfers to and from the data register and other registers can take place on this data bus. A separate Data

Request (DRQ) output is available to synchronise the data transfers through an off-chip DMA controller.

The WD279X floppy disk controllers are designed to interface with only one disk drive. A separate register (described later) was added to interface the controller with four floppy disk drives. These controllers assume that the

drive on which the command is to be executed, is already selected. However, the condition of the 'ready' signal from the drive is monitored before proceeding any further with the command.

4.3.1 The WD279X registers

Unlike the IMDC, WD279X disk controllers are designed as register oriented devices. Communication to and from the controllers is handled through this set of registers. These registers are accessible to the SCC68070 (the CPU) at all times. As described above, there are in all five registers available to the CPU to control the floppy controller functions. Another register, the Mask Register, was added to facilitate interfacing with more than one drive and also to mask out the interrupt for the polling data transfer mode. This was added as a read/write register. The bit definition of the 'mask register' is given in figure 4.13.

The status register is an 8 bit read only register. This reflects the most recent status of the controller. The bit definition of this register depends on the type of command being executed. The status register bit pattern for type I, II and III commands is given in table 4.1. The command register is also an 8 bit register. It is a 'write only' register and holds the 'command byte' for the command be-

ing executed. This register should not be over-written if the controller is busy executing the previous command. A 'force interrupt' command can, however, be written to abort the command in progress. A write to this register automatically starts the command execution. This, therefore, should be the final register written to after filling the other registers with the appropriate information.

The track register holds the track number of the present head position. It is a read/write register and is updated by the controller with every (inward or outward) 'step' of the head movement. The contents of this register are updated by the controller only. The CPU is allowed to read or write to this register but is not allowed to change its contents. Before performing a command, the controller compares the track number on the disk with the contents of this register. If these do not match, the command is aborted. A 'restore to track 0' command can take the controller out of this error situation.

The sector register contains the desired sector number for the read and write operations. It is also an 8 bit read/write register. The contents of the register are matched with the sector number in the ID field of the disk record for the read and write sector operations. If the match is successful, the operation is allowed to proceed. If a match is not found for five disk rotations, the command is aborted with a 'record not found' error.

The data register holds the data for data transfers. For every data transfer, the FDC generates a Data Request (DRQ) signals. This DRQ is reflected in the status register (table 4.1). This register is mainly used for data transfers. However, in the case of a 'seek' command, this register is loaded with the desired track number. It is also an 8 bit read/write register.

The above mentioned registers of the WD279X type controllers are available on the 8 bit data bus. Two address lines, A0 and A1, along with the 'Read Enable' (REN) and 'Write Enable' (WEN) signals can be used to select the desired register. The decoded address for the device and the registers is given in section 4.3.3 (i.e. in the circuit description).

4.3.2 The command procedure

The WD279X floppy disk controllers are capable of executing up to eleven different commands. Each of these commands can be given further flexibility. The internal bits, set within the 'command byte', further enhance the device usage (table 4.2). For example, a 'read sector' command can perform a single sector read if 'm' (bit 4) in the command byte is 'zero'. But it can perform a multiple sector read if 'm' is set to 'one'. Similarly, the verify bit (v = 1) in a seek command switches on an automatic seek verify. In this case, the 'seeked' track number is verified with the track number in the ID field of the track on the disk and the status register is updated accordingly.

The commands for these disk drive controllers can be divided into four types. A summary of command types and their internal flags is included in table 4.2. The type I commands are mostly related to the head movements. The internal flags include the head load flag, a verify flag to enable a track verify at the end, and the bits for programming the step rate. The step rate can be programmed as 6, 12, 20 or 30 mS (at 1 MHz input clock frequency). These type I commands include a 'restore' command (i.e. seek track 00), a command to 'seek' the required track number, 'step' command to move the head one step in the direction specified in the previous command, 'stepin' command and the 'stepout'command.

The type II commands are read/write commands. As these are associated with the head loading and unloading, flags inside the command byte are provided to take care of head load and unload delays and also, in case of WD2795 and WD2797, for the side comparison. The commands in this group are performed on individual sectors. The 'm' flag provides the option for a single or multiple sector operation. In the case of a multiple sector read operation (i.e. to read more than one sectors in one operation), after reading the starting sector with the sector number in the sector register, the sector register is automatically incremented by one and the read starts again. The last sector which will be read successfully will be the the last sector on the track, and the sector number in the sector register will be one more than this. Similarly, for multiple sector write operations, after writing the starting sector the sector register is incremented and the entire operation starts again. These multiple sector operations end with a 'Record Not Found' (RNF) error in the status register (table 4.1). This is because the controller at the end is looking for a sector number higher than the highest sector number on the track. If, for an RNF error, the contents of the sector register are less than or equal to the highest sector number on the track, the error has really occurred, otherwise, the operation was successful.

Type III commands are the same as type II commands other than they are performed on a whole track. The option for multiple operations is not available, but the head loading and unloading delays can be enabled. The commands included in this group are the 'read ID'command, 'read track' command and the 'write track' command. After loading the registers with the appropriate information, the command register is loaded with the command byte and the head is immediately loaded. The operation starts with the first encounter of an index pulse. In the case of a 'read track' command, the controller reads all the gaps, headers and data bytes on the entire track and sends them to the host.

In the case of a 'write track' command, it starts writing data on the disk after receiving the first index pulse. These read and write operations continue until the controller encounters the next index pulse. The write track command is used to format a track. The 'write track' data is arranged in the form of gaps, headers and data and is transferred to the disk exactly in the form of a formatted track. A type IV command is the 'forced interrupt' command. The conditions for an interrupt can be programmed in the command byte to suit the individual requirements (table 4.2). When the programmed condition occurs, the interrupt is generated for the CPU.

The WD279X floppy disk controllers do not have on-chip FIFO type data buffers. Data transfers take place according to the disk drive data rate. The DRQ is generated when the controller is ready for a data transfer. The 'service time' for the DRQ is less than 32 uS for 5.25 inch floppy disk drive (data rate for the drive is 250 KBit/sec). If not serviced within this time, the controller will set the 'data lost' bit in the status register. The command, however, continues. An interrupt is generated at the end of the command. The interrupt is reset either by reading the status register, or by writing a new command byte in the command register.

As mentioned earlier, the controllers can interface to a single floppy disk drive. In the case of multiple drives, separate drive select logic is required. A mask register was added to provide the option to use up to four disk drives. In the event of switching from one drive to another, the track information about the previous drive can be stored separately into the memory and the track number for the new drive can be loaded into the track register. The new drive is enabled through the mask register. In this way commands on more than one drive can be performed smoothly.

4.3.3 Circuit description and the calibration procedure

As described previously, the circuit was designed to accommodate any member of the WD279X family. A set of jumpers was provided for this purpose. The complete circuit diagram is given in figure 4.14. It includes a number of Medium Scale Integrated Circuits (MSIs). The Programmable Logic Device 'PLD' (PAL16L8) was used for the address decoding and for the encoding of REN, WEN and CSN. It also provided the Data Read (DRN) and Data Write (DWN) signals for the 'added' mask register (fig 4.12) and also provided the DTACKN signal required to terminate the SCC68070 bus cycle. The device registers were decoded at the following addresses.

status register = 1FFC00 read only,

command register = 1FFC00 write only,

track register = 1FFC02 read/write,

sector register = 1FFC04 read/write,

data register = 1FFC06 read/write,

mask register = 1FFC08 read/write.

The clock was provided by prescaling a 16 MHz signal. This signal was generated by a 74S124 and 16 MHz crystal while the prescaling was provided by 74LS393, a dual BCD counter. The 74LS164 shift register was used to introduce the required small delays in the circuit. These delays were required to synchronise the speed of the floppy disk controller with the SCC68070. The recommended minimum REN and WEN duration is 200 nS. Since the DTACKN was generated by the PLD

(Appendix D), in the case when it was not delayed, the SCC68070 terminated its bus cycle prematurely and the data read from the disk was not always correct. The 74LS164 was enabled by either UDSN or LDSN. The serial input to this IC was tied high. When clocked at 8 MHz, it produced outputs about 125 nS apart (fig 4.15). DLY2 was used with REN and WEN to ensure that these become active after the CSN signal. The minimum required pulse width for REN and WEN is 200 nS. To ensure this, the DTACKN signal was delayed using DLY4 (i.e. giving REN and WEN a margin of 250 nS).

The mask register (74LS273 and 74LS244) was used to enable or disable the interrupt and the data request (DRQ). It also provided the software selectable single or double density formats. To accommodate those members of the family which do not provide a Side Select Output (SSO), (i.e. WD2791 and WD2793), bit 7 of the mask register was used to enable side '1'. Drive 0 through drive 3 are enabled using bit 3 through bit 6 of the mask register respectively.

A single shot (74LS123) provided the head load delay. It was configured to provide a pulse width of about 60 mS which was the manufacturers' recommended head settling time for the floppy disk drives. Two types of buffers, inverting and non-inverting, were used to interface with the disk drive. This is because the disk controllers had both, positive and negative logic signals. A variable resistance RV1 was connected to the WRW pin (i.e. pin 33). The controller was calibrated to provide the proper write precompensation (180 nS) on the inner tracks. This precompensation was adjusted by pulling the 'TEST' pin (pin 22) to logic 'low' after the reset. RV1 was then adjusted to give the correct pulse width on WD pin (pin 31). To calibrate the data separator, the read pulse width and the on-chip VCO central frequency was adjusted. RV2 was calibrated to give a read pulse width of about 500 nS on the TG43 pin (i.e. pin 29). For the VCO central

frequency adjustment, the VCO output was observed on pin 16 and the VCO central frequency was adjusted by the variable capacitor on the VCO pin (pin 26). It was calibrated to provide a 250 KHz output which is the same as the disk drive data rate.

The circuit was designed for the whole WD279X family. But the WD2793 was actually used. A set of jumpers was provided which could be reconfigured to accommodate the other members. The placement of these jumpers is given in table 4.3.

4.4 Interfacing the DP8474

The DP8474 is one of the family of disk controllers from 'National Semiconductors Ltd'. It is designed as a complete floppy disk controller and requires a minimum of external components. Some of the offered features are:

internal analog data separator,

internal write precompensation,

internal software selectable data rates (125 KBits/sec to 1.25 MBits/sec),

no adjustable components required,

software selectable 'head load' and 'head unload' timing to allow interfacing to 8 inch, 5.25 inch and 3.5 inch floppy drives,

software programmable low power mode (standby current = 100 micro Amps),

extended track range (up to 4096 tracks possible),

can directly interface to up to four floppy disk drives.

This device is designed to fit within a minimal I/O space. It has two registers, the status register and the data register. The status register is a read only register and is available whenever the CPU requires to read the status. The data register is a read/write register and all the communication with the device takes place via this register. Data transfers are also performed via this data register.

The CPU, by reading the status register, can determine the state of the device. When no command is in progress (bit 4 = '0') and the disk controller is ready to receive a byte (bit 7 = '1' and bit 6 = '0'), the command bytes can be written into the data register in the appropriate sequence at a 'controller digestible' speed. The controller can perform an 'implied seek' for the read and write commands. The data to and from the system memory is handled by using 'DMA channel 2' on the SCC68070. For this purpose, the DP8474 provides DRQ (Data Request) and DAK (Data Acknowledge) signals to help synchronise the DMA data transfers. An interrupt is generated by the DP8474 to mark the beginning of the result phase. This interrupt is serviced using the 'auto vector' table on the SCC68070. Therefore, no 'interrupt acknowledge' is required.

4.4.1 The data transfers

The DP8474 has no on-chip data buffer. Therefore, for data transfers, only the data register is available. The data transfers must take place within the 'service time'. This service time for the DP8474 is a function of both the disk drive data rate and the clock frequency. It is given by the following expression:

of the executed command.

The interrupt mode of data transfer is similar to the polled mode. Every time the controller is ready for a byte transfer it generates an interrupt. To make data transfers in this mode, the INT pin on DP8474 should be connected to the CPU and the interrupts would be enabled in the LIR register. The data transfers are to be administered within the 'interrupt service'. The interrupt is cleared after reading a byte from the data register. The same service time restrictions apply for the data transfers. An interrupt is also generated at the beginning of the result phase. Bit 5 in the status register is set if the interrupt was for the data phase and is cleared if it was for the result phase. By reading this bit, the status of the interrupt can be determined and serviced accordingly. This mode of data transfer was not used. The CPU can not handle the FDC interface interrupts for data transfers as fast as it should to avoid the transfer delays. This was because of the multi-tasking nature of 'Tripos'. More than one interrupt can occur at the same time and transfers can be delayed beyond their service time limits.

As mentioned before, the DMA mode can be selected by the 'specify' command. When this mode is selected, each time a data byte is ready to be transferred, a 'DMA request' (DRQ) is generated by the controller. The host (i.e. DMA channel 2 on the SCC68070) responds by the DMA acknowledge (ACK2N) and a Read (RD) or Write (WR) strobes which are decoded by the PLD (Appendix D). At the end of the data phase, an interrupt is generated to mark the beginning of the result phase. This mode of data transfer is much faster compared to the polled mode and implemented for the data transfers.

forms the requested command. The action depends upon the type of command. During this phase (i.e. data phase), the disk controller requires data transfers to and from the data register. Some commands do not require data transfers e.g. 'reset' or 'seek' and therefore have no data phase. During the data phase, if the DMA mode is enabled, the DRQ and DAK signals are used to synchronise the data transfer. Bit 5 in the status register is not set. The service time restrictions always apply for data transfers whether the DMA mode or the polling mode is used. An interrupt is always generated at the end of this phase to mark the beginning of the result phase.

During the result phase, the disk controller, normally, expects a series of bytes to be read from its data register. The interrupt is cleared away after reading the first byte. These bytes along with other relevant information (e.g. the track, side, sector number and the sector code) give the status of the last executed command. If during this phase the controller is not in a 'read from data register' state (i.e. the controller is in the 'write to data register' mode), an 'interrupt sense' would be issued to the disk controller. The result phase outcome of this command will tell the host why the previous interrupt was generated and also (if applicable) the status of the previously executed command which caused that interrupt.

4.4.3 The register map

The DP8474 has two registers, the status register and data register. These registers appear as normal memory locations to the CPU. Both of these registers use the same 8 bit data bus and are separated by a single address line 'A1'. To read the status register A1 should be low and to access the data register it should be high.

The data register is used to write the command bytes and read the result bytes from the controller. It is also used to read and write data to and from the disk drives. This register is 8 bits wide and represents true data only when bit 7 of status register is set. When bit 7 in the status register is not set the data reflected on reading the data register does not represent the true data. This action, (i.e reading the data register when bit 7 in the status register is not set), however, does not upset the controller and it continues as normal.

The second register, the status register, reflects the most recent status of the disk controller. It is a read only register and the contents are always true. The bit pattern of the status register is given below: B7 Request for master, This bit, if set, indicates that the controller requires the attention of the host. This indicates that the controller is ready to send or receive a byte. This bit is cleared after each transfer and is set again for the next one. B6 Data direction, This bit indicates whether the controller is expecting a byte to be written to (B6 = 0) or read from (B6 = 1) the data register. B5 non-DMA execution, This bit is set only during the data phase of the command if the non-DMA mode is selected (by the 'specify' command). If it is set, the data phase is in progress. The data transfers will be handled by the SCC68070 either through the interrupts or by polling. This bit remains cleared during the DMA mode. B4 Busy, This bit is set after the first byte of the command is written, and is cleared after the last byte of the result is read from the data register. B3-B0 Drive seeking, These bits are set for the respective drive for 'seek' or 'restore' commands. It is cleared after reading the first byte of the interrupt sense command for the same drive.

The controller uses the lower half of the microcore data bus (i.e. D0-D7). The byte size data transfers were managed using the 'dual address' data transfer mode of the DMA channel 2 on the SCC68070. Signals like RESET, DRQ, TC and INT were inverted in the same PLD so as to interface directly to the microcore. DAK is an active low signal with the same logic sense as on the SCC68070. This was connected directly without any inversion. Pin 23 on the DP8474 is an output for the 'charge pump' and is also the input to the on-chip VCO. A simple filter comprising R1, C1 and C2 was connected to this pin. Another resistance, R2, was used to set the charge pump current. The values used were R1 = 750 Ohms, R2 = 10 KOhms, C1 = 330 nF and C2 = 1 nF. These were the recommended values for the 250 KBits/sec data rate [40].

The data rate pin, pin 36, was tied high. This allowed a software selectable data rate of between 250 Kbits to 1.0 Mbit per second at an 8 MHz clock frequency. On 'reset' the data rate, as selected by the data rate pin, is 500 Kbits/sec. Afterwards the data rate is changed to 250 Kbit/sec during the 'mode' command. The disk drive interface signals (i.e. drive select, read data, write data etc) are normally positive logic implemented on DP8474. By pulling the 'invert' pin (pin 18) high, these signals become active low. This allows direct interface with the disk drives. The output signals from the disk drives are open collector types. They were terminated with 1.0 KOhms pullups (RP1 in figure 4.16). Similarly, the inputs to the disk drives were terminated with 1.0 KOhms resistances on the disk drives. This was because the output pins (to the disk drives) of the DP8474 can sink up to 8.0 mA. This termination arrangement eliminates any need for extra buffering on the disk drive end of DP8474 and it was directly interfaced to the disk drives. The software to manipulate the FDC interface (i.e. device driver for floppy disk controller) is given in chapter 6.

4.5 Comparison and limitations

As described before, three pieces of the hardware for the FDC interface were built around Large Scale Integrated circuits (LSI). These LSIs were the IMDC, the WD2793 and the DP8474. The aim was to provide the most suitable floppy disk controller.

The IMDC offers very attractive features, such as, the on-chip DMA controller and a choice of direct interface to both floppy disk and hard disk on both ST-506 and SA-1000 standards. With the help of a decoding circuit it can choose between the DPLLs for the hard disk and the floppy disk. The requirement, however, was for a floppy disk controller. For this purpose only, the IMDC needed substantial supporting hardware. The circuit built for this purpose consisted of approximately seventeen ICs (fig 4.4). This was not a small circuit foot print. The increased number of ICs also increased the power consumption which also meant additional cost. The write precompensation also required some additional external hardware. The delay lines were used for this purpose. This also limited the flexibility of the hardware. A software programmable capability was more preferred.

The second piece of hardware contained a WD2793 at the heart of the FDC interface. It made a compact controller with an on-chip data separator and on-chip write precompensation logic. This makes the things a lot easier. The main draw back was the ability to interface with only one drive. A separate arrangement was required to use more than one drive. This was provided in the form of an external register. The overall circuit required fewer ICs (fig 4.14) compared to the circuit built around the IMDC. The calibration procedure proved to be more complicated than with the IMDC.

The third circuit was built around the DP8474. The DP8474 is designed as a stand alone floppy disk controller with the maximum number of programmable functions. The external component requirement is kept to a minimum with just two resistors four capacitors and a crystal (fig 4.16). A PLD provides the device decode and inversion for the positive logic signals. Because proper terminators were used (i.e. 1.0 KOhms), no external current buffers were required to interface to the floppy disk drives. Moreover, the invert pin is available to interface with a positive or negative logic implemented disk drives. The controller circuit consisted of just two ICs, that is the best possible trade off. Because of its compactness and built in facilities, this circuit was chosen as the floppy disk controller interface to be implemented on the I/O controller board.

4.6 Summary

The three circuits, for interfacing to floppy disk drives have been described. These were built around the SCN68454, the WD2793 and the DP8474. The circuit descriptions for all the implemented circuits have been given. At the end a comparison was made between the three FDC interface circuits. The SCN68454 (IMDC) is equipped with an on-chip DMA controller for data transfers. This makes it fast and eliminates the off-chip DMA requirement for data transfers. As the SCC68070 (i.e. the processor used for the I/O purposes) already has two on-chip DMA channels, the DMA onboard the IMDC has no importance. The extra logic required to implement the circuit, the external DPLL and the external VCO counted as drawbacks.

The WD2793 had the on-chip phase lock loop as well as the VCO. Minimal external components were required to trim them to give the required output.

A variable capacitor and a variable resistance were used to calibrate the data separator (on-chip PLL and VCO). The controller, however, interfaces with just one disk drive. This added extra hardware to the circuit. The overall chip count remained less than that required for the IMDC implementation and was therefore, a better choice than the IMDC. But the DP8474 offered something more. It reduced the floppy disk controller implementation to a mere two ICs. The other required components were a few resistors, capacitors and a crystal. Everything could be programmed through the software. Therefore, the DP8474 offered the best possible choice for the given requirement.

The implemented hardware used DMA channel 2 on SCC68070 and the interrupt from the disk controller were serviced from the on-chip autovector table. These interrupts were received on INT1N pin of the SCC68070. The software for the disk controller is given in chapter 6. Before this, in chapter 5, some more interface circuits (e.g. SASI and I2C) are described. This chapter along with the chapter 5 would complete the hardware description on the I/O board.

service time = (8 / data rate) - (16 / clock frequency)

When the disk controller is operating at 8 MHz and the drive data rate was 250 KBits/sec, the service time comes out as 30.0 uS. Therefore, the controller's data request should be serviced within 30.0 uS. If the data is not transferred within this time, the controller will abort the command and set the data overrun error in the 'result status' bytes (table 4.4).

The DP8474 allows disk transfers in either a 'DMA' mode or a 'non-DMA' mode. The non-DMA mode can be either a 'polling' mode or an 'interrupt driven'mode. a DMA or a non-DMA mode is selected by the 'specify' command [40].

In the polling mode, after issuing the command, the status register was polled to see when a data transfer can take place. The interrupts on the CPU were switched off in the LIR (chapter 3). Otherwise, the interrupt pin (INT) of the disk controller should be left open (fig 4.16). This was to avoid interrupts from the disk controller, because, in the non DMA mode, the disk controller generates an interrupt whenever it is ready to make a byte transfer. Also if other peripherals interrupt during the data phase, the service of these interrupts can cause a delay in handling the disk controller data transfers within its service time, causing a data overrun or underrun error.

When a byte is to be read from the data register, bit 6 and 7 should be set in the status register. Similarly, when a byte is to be written to the data register, bit 7 in the status register should be set and bit 6 should be cleared. During this mode of data transfer, bit 5 in the status register can be monitored. If this bit is set, the data phase is in progress, and when it is cleared the result phase has begun. The bytes read during the result phase can be analysed to find the status

of the executed command.

The interrupt mode of data transfer is similar to the polled mode. Every time the controller is ready for a byte transfer it generates an interrupt. To make data transfers in this mode, the INT pin on DP8474 should be connected to the CPU and the interrupts would be enabled in the LIR register. The data transfers are to be administered within the 'interrupt service'. The interrupt is cleared after reading a byte from the data register. The same service time restrictions apply for the data transfers. An interrupt is also generated at the beginning of the result phase. Bit 5 in the status register is set if the interrupt was for the data phase and is cleared if it was for the result phase. By reading this bit, the status of the interrupt can be determined and serviced accordingly. This mode of data transfer was not used. The CPU can not handle the FDC interface interrupts for data transfers as fast as it should to avoid the transfer delays. This was because of the multi-tasking nature of 'Tripos'. More than one interrupt can occur at the same time and transfers can be delayed beyond their service time limits.

As mentioned before, the DMA mode can be selected by the 'specify' command. When this mode is selected, each time a data byte is ready to be transferred, a 'DMA request' (DRQ) is generated by the controller. The host (i.e. DMA channel 2 on the SCC68070) responds by the DMA acknowledge (ACK2N) and a Read (RD) or Write (WR) strobes which are decoded by the PLD (Appendix D). At the end of the data phase, an interrupt is generated to mark the beginning of the result phase. This mode of data transfer is much faster compared to the polled mode and implemented for the data transfers.

4.4.2 The command procedure

The DP8474 performs a number of commands. The operations performed, depend on the type of the issued command. Before receiving any command, the controller is in the 'idle' state. In this state the disk drives are continuously polled and the 'ready' signal from the drives is monitored. If it changes state, an interrupt is generated. This drive polling mode is disabled by the 'mode' command. The polling mode is also disabled automatically if the controller is not in the idle state.

Before issuing a command, the controller should be ready to receive a byte (i.e. bit 7 in status register is set and bit 6 is cleared). If the controller is not in this state, then it must be forced into this state before issuing any further command bytes. This is done by force issuing an illegal command (i.e. a command which is not included in the list of controller commands [40]). Alternatively, an 'interrupt sense' command can be used to bring the controller into the ready state.

The 'command procedure' for the DP8474 has three phases. These three phases can be named as the command phase, the data phase and the result phase. During the command phase a number of command bytes are written into the data register. These bytes have a predefined structure and contain the information and parameters required to execute the command [40]. These bytes are written into the data register in the specified order. The command phase continues until the last byte of the command has been written into the data register. During this phase no interrupt or DMA data requests are generated, therefore, 'service time' restrictions do not apply.

The command phase is followed by an immediate data phase. The controller per-

forms the requested command. The action depends upon the type of command. During this phase (i.e. data phase), the disk controller requires data transfers to and from the data register. Some commands do not require data transfers e.g. 'reset' or 'seek' and therefore have no data phase. During the data phase, if the DMA mode is enabled, the DRQ and DAK signals are used to synchronise the data transfer. Bit 5 in the status register is not set. The service time restrictions always apply for data transfers whether the DMA mode or the polling mode is used. An interrupt is always generated at the end of this phase to mark the beginning of the result phase.

During the result phase, the disk controller, normally, expects a series of bytes to be read from its data register. The interrupt is cleared away after reading the first byte. These bytes along with other relevant information (e.g. the track, side, sector number and the sector code) give the status of the last executed command. If during this phase the controller is not in a 'read from data register' state (i.e. the controller is in the 'write to data register' mode), an 'interrupt sense' would be issued to the disk controller. The result phase outcome of this command will tell the host why the previous interrupt was generated and also (if applicable) the status of the previously executed command which caused that interrupt.

4.4.3 The register map

The DP8474 has two registers, the status register and data register. These registers appear as normal memory locations to the CPU. Both of these registers use the same 8 bit data bus and are separated by a single address line 'A1'. To read the status register A1 should be low and to access the data register it should be high.

The data register is used to write the command bytes and read the result bytes from the controller. It is also used to read and write data to and from the disk drives. This register is 8 bits wide and represents true data only when bit 7 of status register is set. When bit 7 in the status register is not set the data reflected on reading the data register does not represent the true data. This action, (i.e reading the data register when bit 7 in the status register is not set), however, does not upset the controller and it continues as normal.

The second register, the status register, reflects the most recent status of the disk controller. It is a read only register and the contents are always true. The bit pattern of the status register is given below: B7 Request for master, This bit, if set, indicates that the controller requires the attention of the host. This indicates that the controller is ready to send or receive a byte. This bit is cleared after each transfer and is set again for the next one. B6 Data direction, This bit indicates whether the controller is expecting a byte to be written to (B6 = 0) or read from (B6 = 1) the data register. B5 non-DMA execution, This bit is set only during the data phase of the command if the non-DMA mode is selected (by the 'specify' command). If it is set, the data phase is in progress. The data transfers will be handled by the SCC68070 either through the interrupts or by polling. This bit remains cleared during the DMA mode. B4 Busy, This bit is set after the first byte of the command is written, and is cleared after the last byte of the result is read from the data register. B3-B0 Drive seeking, These bits are set for the respective drive for 'seek' or 'restore' commands. It is cleared after reading the first byte of the interrupt sense command for the same drive.

4.4.4 Hardware considerations and the circuit description

The DP8474 is designed to be a 'complete' floppy disk controller. It requires a minimum of external support hardware. In the implemented circuit two ICs were used (fig 4.16), the DP8474 and a PLD (i.e. Programmable Logic Device). The PLD was used for the address decoding and encoding of RD and WR signals (Appendix D). It also generates the DTACKN signal on behalf of the DP8474 to terminate the SCC68070 bus cycle. Some signals like RESET, INT, DRQ and TC (Task Complete) needed inversion. This inversion was required because of the positive logic implementation of these signals on the DP8474. The on-chip oscillator uses an 8 MHz crystal connected externally to pins 20 and 21 on the DP8474 (fig 4.16). This provides the base clock required to synchronise the internal functions of the disk controller.

The PLD decodes the DP8474 registers as:

status register = 1FFC01, and

data register = 1FFC03.

Only two address lines, A8 and A9 were used in the decode circuit. This was done to keep the circuit to the minimum possible chip count. If all the address lines in the I/O area of the microcore board were used (to avoid the possibility of address overlap in this area), an extra chip would be required. Because of the available on-chip peripherals, the use of two address lines gave enough flexibility to attach other required devices in this area.

The controller uses the lower half of the microcore data bus (i.e. D0-D7). The byte size data transfers were managed using the 'dual address' data transfer mode of the DMA channel 2 on the SCC68070. Signals like RESET, DRQ, TC and INT were inverted in the same PLD so as to interface directly to the microcore. DAK is an active low signal with the same logic sense as on the SCC68070. This was connected directly without any inversion. Pin 23 on the DP8474 is an output for the 'charge pump' and is also the input to the on-chip VCO. A simple filter comprising R1, C1 and C2 was connected to this pin. Another resistance, R2, was used to set the charge pump current. The values used were R1 = 750 Ohms, R2 = 10 KOhms, C1 = 330 nF and C2 = 1 nF. These were the recommended values for the 250 KBits/sec data rate [40].

The data rate pin, pin 36, was tied high. This allowed a software selectable data rate of between 250 Kbits to 1.0 Mbit per second at an 8 MHz clock frequency. On 'reset' the data rate, as selected by the data rate pin, is 500 Kbits/sec. Afterwards the data rate is changed to 250 Kbit/sec during the 'mode' command. The disk drive interface signals (i.e. drive select, read data, write data etc) are normally positive logic implemented on DP8474. By pulling the 'invert' pin (pin 18) high, these signals become active low. This allows direct interface with the disk drives. The output signals from the disk drives are open collector types. They were terminated with 1.0 KOhms pullups (RP1 in figure 4.16). Similarly, the inputs to the disk drives were terminated with 1.0 KOhms resistances on the disk drives. This was because the output pins (to the disk drives) of the DP8474 can sink up to 8.0 mA. This termination arrangement eliminates any need for extra buffering on the disk drive end of DP8474 and it was directly interfaced to the disk drives. The software to manipulate the FDC interface (i.e. device driver for floppy disk controller) is given in chapter 6.

4.5 Comparison and limitations

As described before, three pieces of the hardware for the FDC interface were built around Large Scale Integrated circuits (LSI). These LSIs were the IMDC, the WD2793 and the DP8474. The aim was to provide the most suitable floppy disk controller.

The IMDC offers very attractive features, such as, the on-chip DMA controller and a choice of direct interface to both floppy disk and hard disk on both ST-506 and SA-1000 standards. With the help of a decoding circuit it can choose between the DPLLs for the hard disk and the floppy disk. The requirement, however, was for a floppy disk controller. For this purpose only, the IMDC needed substantial supporting hardware. The circuit built for this purpose consisted of approximately seventeen ICs (fig 4.4). This was not a small circuit foot print. The increased number of ICs also increased the power consumption which also meant additional cost. The write precompensation also required some additional external hardware. The delay lines were used for this purpose. This also limited the flexibility of the hardware. A software programmable capability was more preferred.

The second piece of hardware contained a WD2793 at the heart of the FDC interface. It made a compact controller with an on-chip data separator and on-chip write precompensation logic. This makes the things a lot easier. The main draw back was the ability to interface with only one drive. A separate arrangement was required to use more than one drive. This was provided in the form of an external register. The overall circuit required fewer ICs (fig 4.14) compared to the circuit built around the IMDC. The calibration procedure proved to be more complicated than with the IMDC.

The third circuit was built around the DP8474. The DP8474 is designed as a stand alone floppy disk controller with the maximum number of programmable functions. The external component requirement is kept to a minimum with just two resistors four capacitors and a crystal (fig 4.16). A PLD provides the device decode and inversion for the positive logic signals. Because proper terminators were used (i.e. 1.0 KOhms), no external current buffers were required to interface to the floppy disk drives. Moreover, the invert pin is available to interface with a positive or negative logic implemented disk drives. The controller circuit consisted of just two ICs, that is the best possible trade off. Because of its compactness and built in facilities, this circuit was chosen as the floppy disk controller interface to be implemented on the I/O controller board.

4.6 Summary

The three circuits, for interfacing to floppy disk drives have been described. These were built around the SCN68454, the WD2793 and the DP8474. The circuit descriptions for all the implemented circuits have been given. At the end a comparison was made between the three FDC interface circuits. The SCN68454 (IMDC) is equipped with an on-chip DMA controller for data transfers. This makes it fast and eliminates the off-chip DMA requirement for data transfers. As the SCC68070 (i.e. the processor used for the I/O purposes) already has two on-chip DMA channels, the DMA onboard the IMDC has no importance. The extra logic required to implement the circuit, the external DPLL and the external VCO counted as drawbacks.

The WD2793 had the on-chip phase lock loop as well as the VCO. Minimal external components were required to trim them to give the required output.

A variable capacitor and a variable resistance were used to calibrate the data separator (on-chip PLL and VCO). The controller, however, interfaces with just one disk drive. This added extra hardware to the circuit. The overall chip count remained less than that required for the IMDC implementation and was therefore, a better choice than the IMDC. But the DP8474 offered something more. It reduced the floppy disk controller implementation to a mere two ICs. The other required components were a few resistors, capacitors and a crystal. Everything could be programmed through the software. Therefore, the DP8474 offered the best possible choice for the given requirement.

The implemented hardware used DMA channel 2 on SCC68070 and the interrupt from the disk controller were serviced from the on-chip autovector table. These interrupts were received on INT1N pin of the SCC68070. The software for the disk controller is given in chapter 6. Before this, in chapter 5, some more interface circuits (e.g. SASI and I2C) are described. This chapter along with the chapter 5 would complete the hardware description on the I/O board.

Chapter 5

SASI and I2C interface

5.1 Introduction

In the previous chapter, the circuits to interface floppy disk drives to the SCC68070 were described. This chapter describes the interface to another magnetic disk storage peripheral; the hard disk. Hard disks are fast devices. They have faster access times compared to floppy disk drives and offer larger storage capacities. These disks are normally available with a SCSI interface fitted to them or available as a separate module. The SASI bus is a subset of the SCSI bus. This chapter describes the SCSI bus requirements and the interface circuit which was designed to interface the SCC68070 to the SASI bus and hence to the hard disk.

Once described as Shugart Associates Systems Interface [42], the SASI bus was later modified and standardised as the Small Computer Systems Interface (SCSI). It is a parallel bus and offers a relatively high data transfer rate of up to approximately 4 Mbytes/sec. It uses inexpensive components (drivers, receivers, cables, connectors etc), and operates over moderate distances (up to 25 metres). The command set is device independent, that is, it normally does not disclose the internal structure of the device (e.g. heads, cylinders etc in case of a hard disk),

thus allowing the same software to control all devices of a particular type. The SASI bus, being a subset of the SCSI bus offers all these capabilities. The only exception is the 'message system' other than the 'command complete' or some 'intermediate' messages [43]. The SASI bus can not support the 'selection' and 'reselection' procedures and, therefore, it can not support the command independent messages. These are also not included in the discussion.

The second bus discussed in this chapter is the Inter-IC (I2C) bus [44]. This is a serial bus and operates independent of the centralised bus orbiter. This is a two wire bus, comprising of the 'Serial Clock' (SCL) and 'Serial Data' (SDA) lines. It is normally used to perform remote control functions which do not require very high speed data transfers. This bus was used to connect SCC68070 to a real time clock and also an I/O expander to provide a parallel centronics interface for the printer.

The link adapter circuit was added to the I/O board to facilitate communication with the rest of the transputer based computer. It is also included to complete the discussion about the hardware of the I/O board.

5.2 The SCSI bus

The SCSI bus allows the host computer expansion and interface capabilities to certain semi-intelligent independent devices. The bus specification provides the possibility of connecting these devices independently to the host without making any generic changes in hardware and software. Devices connected on this bus recognise the standard SCSI commands and are capable of executing these commands independently. For example, if the hard disk is used on this interface and

the host issues a 'read block' command, it does not need to tell the device how and where to locate the required block. The required block number is sent to the disk device and the disk device itself with the help of its SCSI interface is capable of determining the surface, track and the sector number. Moreover, the command structure remains the same for all SCSI devices.

SCSI is a small localised bus used mainly for connecting peripherals to the computers. These peripherals include disk devices, tape devices, printers, plotters, laser printers etc. The bus is strictly standardised for small lengths (a maximum of 25 metres) and can not be used for local area networks. This parallel data bus can support a high data transfer rate. Connected devices are termed as the 'initiator' or the 'target'. An initiator is a device which initiates the command. It is normally the host computer. The target is the device which performs the issued command (e.g. disk device). Once the command is issued, the initiator and the target normally can not change their roles. But, sometimes, if the target possesses the capability, it can change its role to an initiator. A good example is the 'copy' command. Devices which can support this command switch their status and become an initiator to execute the command on another target device. The bus can support a 'single initiator single target', 'single initiator multiple target' system (fig 5.1) or a 'multiple initiator multiple target' system (fig 5.2).

5.2.1 The SCSI bus signals

The SCSI bus can support either a single ended or differential ended system. The pin connections for both cases are given in figure 5.3. The signals are active low and have standard termination for both single ended and differential outputs (fig 5.4). The bus signals are given below.

D0-D7 8 bit data bus. The data is defined true when the corresponding bit is driven low. D0 is the least significant bit and D7 is the most significant bit. It is used to transfer command, data, result and message bytes between the computer and the device.

DB(P) The parity bit. The parity on the SCSI bus is supported as 'odd' and is optional. Devices are not necessarily required to support the parity bit and is sometimes not implemented

Terminator Power This is also optional. It is a power pin and if the option is provided it has the following requirements. Single ended: +4.0 Vdc to +5.25 Vdc 800 mA minimum source drive capability, 1.0 mA maximum sink capability. (recommended fuse 1.0 Amp). Differential: +4.0 Vdc to +5.25 Vdc 600 mA minimum source drive capability, 1.0 mA maximum sink capability.

(recommended fuse 1.0 Amps).

ATN Attention. Active low signal wire Or'd type (if multiple initiator system). It is driven by the initiator to indicate the attention condition (for message passing).

BSY Busy. A wire Or'd signal, driven by the target to tell other devices that the bus is occupied and being used. It is driven low after the device selection procedure and remains low until the result phase is complete.

ACK Acknowledge. A signal driven low by the initiator to acknowledge the REQ/ACK transfer handshake for data transfers.

RST Reset. A wire Or'd signal driven low to impose a reset condition on all the SCSI devices. It overrides all phases and starts a reset condition. The bus becomes 'idle'.

MSG Message. This is driven by the target during the message phase.

SEL Select. This signal is driven by the initiator to select the target or driven by the target to reselect the initiator if the initiator and target can support disconnection.

C/D Control/Data. This is driven by the target to tell the initiator that the control (low) or data (high) byte is or should be presented on the data bus.

REQ Request. This is driven by the target to request a byte transfer. The information is transferred using the REQ/ACK handshake.

I/O Input/Output. This signal is also driven by the target to determine and control the direction of transfer. A high indicates the transfer is to the target and a low indicates that the transfer is to the host.

The minimum requirements for a SCSI bus are at least one initiator and one target. It can directly support up to eight SCSI devices. To start the communication with the target the host or the initiator goes through the bus arbitration procedure. This procedure is discussed in the following section.

5.2.2 The SCSI bus phases

After reset the SCSI bus is forced into the idle state. All the signals on the bus are either tristated or go into their inactive state and the connected target devices start their startup (or initialisation) procedures. After the devices have initialised themselves, they stay idle until the initiator starts the device selection procedure.

When the initiator wants to issue a command to a target device, it waits for the

bus to become available. After detecting that 'BSY' is not low (i.e. no other device is using the bus), it places the device's SCSI address on the data bus and asserts the 'SEL' signal (fig 5.5). The addressed target device, after recognising the call, responds and occupies the bus by asserting the BSY signal. The initiator then removes the 'address' and the 'SEL' signals from the bus. The target then asserts the 'C/D' signal to request a command phase to the initiator. The 'BSY' signal remains asserted as long as the bus remains busy, i.e. during the command, data and result phases.

The command phase is the first phase within the 'bus busy' phase. On assertion of the 'REQ' signal from the target after the selection sequence is complete, the initiator places the first byte of the command field on the data bus and asserts the 'ACK' signal. The target device, after reading the byte, removes the 'REQ' signal and waits for the 'ACK' to become inactive before making the next data request (fig 5.6). This handshake continues until all the command bytes are transferred.

The command phase is followed by the data phase. Some commands do not have the data phase (e.g. the restore command). But if the data phase is applicable, the C/D signal becomes deasserted to indicate this. The I/O signal indicates the data direction (a high for data to the target and low for data from the target) and the REQ/ACK handshake assures a correct data transfer. In the case of data from the target, data is valid when REQ becomes valid. Similarly, the initiator in the 'data to the target' phase, expects valid data when the 'ACK' signal is asserted (fig 5.7). The BSY signal remains valid and the handshake is required for each single byte transfer.

Following the data phase, the status phase is used to send a status byte to the initiator to provide information about the previous command. The C/D signal is

asserted to mark the beginning of the status phase. If the command was aborted by the 'abort' message or by a device reset message, then, the status byte is not transferred. Similarly if a 'reset' is applied, the status phase does not exist. The status byte code values are given in table 5.1. A 'good' status means the command was successful and terminated normally. The data transferred during the data phase is valid. A 'check' condition in the status byte means an abnormal termination of a command. Further investigation by using the 'request sense' command can reveal the nature of the fault. A 'busy' condition means that the target was unable to accept the command because of some other 'occupation' and the initiator should wait until some later time when the target becomes available. This target occupation can be with another initiator for a multiple initiator system. If no other initiator is involved, then, a hardware fault could be causing this 'busy' condition.

If linked commands are used, then the target returns an intermediate status byte at the end of each command and starts the next command. If some error occurs, the command chain is broken and the full status is returned. No further command in the chain is executed and a 'request sense' command for a 'check condition' status (as would be in this case) can give the details as to why the last command failed.

The 'status' byte is followed by the message byte. A completion message is transferred to indicate that the operation is complete. To transfer a message byte, the target will assert the 'MSG' signal along with the C/D and the I/O signal and places the message byte on the data bus. The 'REQ' signal is then asserted (fig 5.8). This is the last byte to be transferred to complete a command execution. The target then releases the bus by negating the 'I/O', 'C/D' and 'BSY' signals. It enters the 'idle' state and waits for the next 'bus arbitration

and selection' phase.

5.2.3 The SCSI commands

SCSI devices at the interface are assumed as a set of continuous logical blocks of a predefined length. This block length is either fixed and known to both the initiator and the target, or it is explicitly defined in the command (e.g. the FORMAT command). A single SCSI command can transfer one or more logical blocks. Commands can also be linked to form a chain for execution purposes provided these are for the same target device.

The SCSI commands are divided into eight groups (groups 0 - 7) [43]. The 'group 0' commands are six byte commands while 'group 1' and 'group 5' are of ten and twelve bytes length respectively. These commands (i.e. group 0, 1 and 5) are standard SCSI commands. Group 2, 3 and 4 commands are reserved for future expansion while group 6 and 7 are vendor unique commands. During the command phase, the initiator makes an 'action request' to the target. This is done by sending a 'command descriptor block' to it. A typical command descriptor block consists of command type along with the information required to execute that command (fig 5.9). The 'Operation Code' (Opcode) is a combination of command code and group code. This gives both the command size and the action. The logical unit number field provides the capability of addressing up to eight physical or virtual devices using the same SCSI address. This increases the total addressing range on the SCSI bus from eight devices to sixty four, provided the attached devices are arranged as having different logical unit numbers. The relative address bit is used only in group 1 and group 5 commands. A 'one' in this field indicates that the number given in the logical block number field is actually

a 2's complement displacement and is to be added to the previously accessed block number to calculate the actual physical block number for this command.

The logical block address is a 21 bit binary number for the six byte commands and is a 32 bit number for ten and twelve byte commands. This normally gives the 'starting block number' for the command. It starts from 'zero' and continues until the highest block number on the accessed device. The size of the block, as previously mentioned, is known to both the initiator and the target. Otherwise, it may be pre-established before initiating data transfers [43]. The total number of blocks to be transferred is given in the 'transfer length' field of the command descriptor block. This number is usually between '1' and '255'. The target transfers only the specified number of blocks. A 'zero' in this field means that a total of '256' blocks will be transferred. In some commands this field also means the number of bytes to be transferred, for example the 'request sense' command [43].

The last byte of the command descriptor block is the 'control byte'. Bit '0' is used for the linked commands. If this bit is set, the target assumes that the next command is a linked command to the previous one. It will return the intermediate status byte followed by the intermediate message byte and then starts looking for the next command. For the message byte, however, it will look for the flag bit (bit 1). If it is set, it will return a 'linked command complete' message with flag (i.e. '0BH') and if it is cleared, it will return a 'linked command complete' message (i.e. '0AH') [43].

5.2.4 The SASI interface circuit design

The SASI bus is a subset of the SCSI bus. The ATN signal in the SCSI bus is not included in the SASI bus (fig 5.3). This implies that, in the SASI bus, the special attention condition can not be forced and the 'attention message' phase can not be introduced. These attention messages are, therefore, not described here. The rest of the specifications for the SASI bus remain the same as for the SCSI bus. An initiator or a target, sometimes, cannot recognise all the approved SCSI commands, therefore a command can be used only if it is supported by both, i.e. initiator and the target. The bus phases remain same for SASI as for SCSI and also there is no difference between the data transfer handshake or the bus arbitration procedures. The command descriptor block and the command bytes themselves remain the same and no overhead is involved. The SASI bus was chosen for the interface design as only a hard disk drive and a tape streamer device were to be attached on this bus. Even if a more intelligent device is attached later, the SASI interface should prove sufficient as the attached device can interrupt the SCC68070 to start a pseudo-attention condition.

The interface was designed to transfer data by the polling mode, the interrupt transfer mode or by using the DMA controller. The SCC68070's DMA controller channel 1 was used. This channel only provides the option of single address data transfers. Therefore, the provisions were made to interface the 16 bit SCC68070 bus to the 8 bit SASI bus for single address data transfers. The complete circuit diagram of the interface is given in figure 5.10.

The interface circuit was designed with three registers, the status register, the mask register and the data register. These registers were decoded at the following addresses:

status register = 1FFD01 read only,

mask register = 1FFD01 write only,

data register = 1FFD03 read/write.

The status register is a five bit wide register and can be read to monitor the state of the SASI bus signals (table 5.2). If the bus is idle, it returns a '1FH' status value. If a command is in progress, the 'status' depends upon the situation present. The mask register is a 7 bit wide register (table 5.2.3). It can be used to force the 'SEL' and 'RST' signals on the SASI bus. It is also used to enable or disable DMA requests and interrupt to the CPU. The data register provided a path to transfer data to and from the SASI bus.

As shown in the circuit diagram the two ICs, R1 and R2 are used to interface the 8 bit SASI bus to the 16 bit SCC68070 bus. These registers (R1 and R2) latch the data on each half of the bus before it is transferred. For example, when the data is transferred to the SCC68070, the target device first puts data on the 'SASI data bus' (SD0-SD7) and then asserts 'REQ'. The REQ signal latches the data in both R1 and R2. If the DMA request was enabled in the mask register (Q2 = 1), the REQ signal also clocks the 'F2' flip-flop. The QN output of this flip-flop is used to make 'Data Request' (DREQ) to the DMA. After this, when the DMA starts the 'write to memory' cycle, both R1 and R2 are enabled. Data transfer to memory takes place according to the state of the LDSN and UDSN signals. Similarly, during a 'read from memory' cycle, the decoded 'WDATA' signal latched data in both R1 and R2. This time the UDSN signal is also latches in the 'F1' flip-flop. The latched output of this flip-flop (LUDSN) is used to enable either R1 or R2 during the 'SASI read' cycle. This arrangement ensures that the correct byte (either from R1 or from R2) is transferred to the SASI bus. The REQ signal

also enables the F3 flip-flop. This flip-flop is clocked at the end of the SCC68070 bus cycle (fig 5.11). The output of this flip-flop acknowledges the SASI request. This terminates the SASI bus cycle. The SASI bus is acknowledged at the end of the SCC68070 bus cycle to avoid the next REQ breaking into the unfinished SCC68070 bus cycle.

The other two registers are R3 and R4. R3 is an octal latch. It holds the mask byte from the CPU. The status register, R4, is simply a set of buffers. When enabled by a decoded signal (STATUS), the output of these buffers reflects the correct state of the SASI bus signals.

After testing this circuit with a hard disk (Rodime's RO650 [47]. the designed circuit was programmed into a PLHS501, the Programmable Logic Device (PLD) [34]. The PLD listing is included in Appendix D. Initially the externally connected flip-flops 'Freq' and 'Fack' (fig 5.12) were fixed inside the PLD. But, because the PLD outputs were found to be unstable, these flip/ flops were taken out of the PLD and connected externally. The full circuit which was initially implemented as three ICs ended up as seven ICs. However, this can be reduced to three ICs later when the internal faults in the PLD are rectified.

5.2.5 The command execution procedure

The designed SASI interface offers three possible modes of data transfer; the polling mode, the interrupt driven mode and the DMA mode. In the polling mode, the CPU continuously polls the status register. The DMA request and the interrupt requests can be disabled through the mask register. After selecting the device (section 5.2.2), the status register is polled and on REQ the command

bytes are written to the data register. Writing to the data register generates an acknowledge to the SASI device as well. After the command phase is complete, the CPU continues to poll the status register for data transfers and the status and message bytes at the end of the command.

The interface controller can also be programmed to interrupt the SCC68070 as soon as the device is ready to transfer a byte. For this purpose the DMA request must be switched off in the mask register and, after the device selection procedure, the interrupt for both the command phase and the data phase can be enabled by putting '78H' in the mask register. The interface controller will then generate an interrupt for each of the command bytes, the data bytes and also for the status and message bytes at the end of the command. On an interrupt, the CPU will write a command byte if the device is in a command phase or transfer a data byte if it is in the data phase. The transfer for a command phase or a data phase is decided by reading the C/D signal from the status register. It will be low for the command phase and high for the data phase. A low C/D signal after the data phase indicates the result phase. In this phase the CPU should read the status and message bytes from the SASI device and the command will end.

The DMA mode can be programmed to make data transfers using the DMA in single cycle mode. After selecting the device, the command bytes are sent to it by writing them to the data register using the polling mode. DMA requests are enabled at this stage and also the interrupt is enabled for the command phase ('74H' in the mask register). A DMA request will be generated for each data transfer request and an interrupt will be generated at the end of the data phase. On interrupt the CPU can read the status and message bytes from the device and, hence, end the SASI command procedure.

In the SCC68070 two DMA channels are available, DMA channel 1 and DMA channel 2 (chapter 3). Channel 1 can be used only in the single cycle mode and has a higher priority than channel 2. Channel 2 can be used in single cycle as well as in dual cycle mode. In the present system it was necessary to use the channel 2 for the floppy disk interface as that circuit could only work with the dual cycle DMA. This made it necessary to use the single cycle DMA for the SASI interface, using channel 1. At present, only the hard disk is used on the SASI bus. This causes problems when both hard and floppy disk drives are used simultaneously. The higher priority hard disk DMA channel locks out the lower priority floppy disk DMA channel. However, data transfers from the floppy disk are time critical. Therefore, it was necessary to overcome this problem.

A 'Test And Set' (TAS) semaphore scheme was used to solve this problem. Using a shared flag, both hard and floppy disk drives are made to adopt a handshaking procedure. The interrupts on the SCC68070 are switched off and each channel waits for the other to finish (i.e. it waits until the flag is cleared). After this, the local channel sets the flag, switches on the interrupts and proceeds with data transfer. After finishing the data transfer, it clears the flag to let the other channel proceed with it's data transfer.

This problem arises because of the multi-tasking nature of the 'Tripos' [46]. If a single task operating system was used this would not have occurred. The above mentioned arrangement meant that effectively one DMA channel was making data transfers at any one time. This makes it slower compared to when two channels are working together and make transfer data simultaneously. The situation arises only when the hard disk and the floppy disk are used simultaneously. Because the two disks are very rarely used together, this would not matter much. In normal situations, when either the hard disk or the floppy disk are being used,

data transfer can occur at the full DMA transfer rate.

5.3 The I2C interface

The I2C (Inter-Integrated Circuit) bus [45] is a serial bus used for 8 bit applications. It transfers serial data on a two wire system. These wires are named the 'Serial Data' (SDA) and the 'Serial Clock' (SCL) lines. Being a serial bus, it does not offer the throughput of a parallel bus, but requires fewer wires and is subsequently cheaper. The two lines (SDA and SCL) are bidirectional lines pulled up using external resistors so that they always remain high when the bus is free.

Devices connected to the bus are either transmitters or receiver at any one time. A transmitter is a device which is placing data on the bus while a receiver is a device which is collecting data from the bus. These devices also have a second role. They can be either a master device or a slave device. The master initiates transfers, it also generates clock signals and terminates transfers. One clock pulse has to be generated for each data bit transferred through this bus. The maximum allowable transfer rate is 100 Kbits per second, and therefore the maximum clock rate is 100 KHz.

5.3.1 I2C data transfers

Data transmission on the I2C bus begins with a 'start' condition and is terminated by a 'stop' condition (fig 5.13). Only for these conditions does the status of the SDA line change during the 'high clock' period. During the data phase, the SDA

remains valid for the 'high clock' period and can change state only during the 'low clock' periods (fig 5.14). The state of the SDA line in the data phase is controlled by the transmitter, while the 'start' and 'stop' conditions and the SCL line is controlled by the master.

Data bytes transmitted on this bus are 8 bits long and are followed by an acknowledgment from the receiver. To acknowledge a successfully received byte, the receiver sends back a 'zero' bit (fig 5.15). If the receiver fails to acknowledge the transmitted byte then data is considered 'lost'. In this case, the master aborts the transfer, generates a 'stop' condition and restarts the procedure again.

If transmission is successful and the slave device was acting as a receiver, it generates an acknowledge bit. The master uses the 'stop' condition to terminate the data transfer. If the master was acting as a receiver, then it does not acknowledge the last byte, but instead generates the 'stop' condition and terminates the transfer.

5.3.2 The I2C bus arbitration

As mentioned previously, each device connected to the I2C bus has two signals to account for, the SCL and the SDA. These lines are 'open collector' types and are wire AND connected. Therefore, if more than one device is driving the lines, the output is dominated by the device which is driving it low. Considering the SCL line, for example, if more than one masters are driving it at the same time, the clock low period is determined by the master which has the 'longest' low period (fig 5.16). Similarly, the clock high period is determined by the master with the 'shortest' high period.

The SDA line is also wire AND connected. Therefore the logical state of this line, if more than one transmitter is involved, is also controlled by the device which is transmitting the 'low' level. When multiple devices are transmitting, they can continue transmitting (with the synchronised SCL) as long as they are transmitting the same data. The device which transmits a 'high' when the other is transmitting 'low', will loose the bus arbitration (fig 5.17). This device will leave the bus and will switch itself to the receiver mode immediately, as it is possible that the winning master may be trying to address it. If two masters are trying to address the same slave, since the address will be the same, the arbitration can go through the data phase until some data comparison occurs. Since all the masters are transmitting as long as the data is the same, no information is lost, and there is no visible effect on the data transfer to the winning master. Also as the control of the I2C bus is decided by the comparison of data among competing masters, there is no central master and hence there is no order of priority on bus acquisition.

The SCL line determines the data transfer rate. The low clock periods can be extended by the receiver (if possible) if it can not cope with the fast data transfer rate of the master. This provides the handshake for slow I2C devices. Again, on the byte level, if a receiving device can receive data at the same rate as the transmitter, after receiving the data, the receiver can hold the SCL line low (to make its internal 'slow' data transfers) and, hence, force the master into a wait state. In this way the speed of any master can adapt to the receiver speed.

5.3.3 I2C communication format

After the first 'start' condition, the master transmits the address byte of the device it wishes to talk to. The address byte (fig 5.18) consists of the 7 bit slave address with the 'Least Significant Bit' (LSB) location used for the 'Read/Write' request ('1' for a read request and '0' for the write request). The slave responds with the 'acknowledge' bit. The data transfer then takes place. The first of three possible data transfer conditions is a master transmitting to a slave receiver. This is a 'write' request and after addressing the slave, the master continues with the data bytes (fig 5.19a). The SDA direction remains unchanged.

The second possible condition is a master attempting to read information from a slave. In this case the master transmits the first byte, 'the address byte', with the read request. At the first acknowledge the master changes from transmit to the receive mode, and the addressed slave device becomes the transmitter (fig 5.19b). The master generates an acknowledge for each data byte it receives. After the last data byte is received, the master does not acknowledge and ends the communication with the 'stop' condition.

The third condition is that during communication, after sending some data bytes the master needs to change the direction of the data transfer (fig 5.19c). To do this the master starts the communication by sending the first byte as a slave address with the required read or write request. It continues with the same read or write action for the required number of bytes before changing the data direction. To switch the read/write modes, it generates the start condition once more followed by the slave address byte with the new (reversed) read/write request bit. Communication continues with an acknowledgment from the new receiver. This format (i.e. Combined Format of Data Transmission) is equivalent to two read

and write requests following each other with the 'stop' condition missing between them.

Sometimes, it is necessary that a master on the I2C bus talks to all the devices on the bus. A general call is used to address all devices present on the I2C bus. A '00H' byte is transmitted on the bus. It may not be true that every device requires some information from the 'general call' and also it is not necessarily true that every device is equipped to recognise this call. If a device requires some information and can recognise this call, it responds with an acknowledge. The master, then, transmits a second byte. This byte specifies the reason for the general call. The LSB of this second byte is either '0' or '1' (fig 5.20a). If it is zero then this second byte has the following meanings:

'06H' Reset and Reprogram. All the devices reset themselves and rewrite their programmable parts from the programming master or from the hardware.

'02H' Do not Reset but Reprogram address. A reset of devices is not performed. However, the programmable parts of the slave addresses are acquired from the master by software(fig 5.20b).

'04H' Do not Reset but reprogram address. A reset of devices is not performed. However, the programmable parts of the slave address are acquired and programmed from hardware.

The programming master is usually the master initiating the general call. Depending upon conditions, the call is followed by a number of bytes for slave programming (fig 5.20b). The slaves receive these bytes and reprogram themselves.

If the LSB of the second byte is '1', then the general call is a hardware call. A call from a master who does not know whom to talk to. The rest of the byte (i.e. 7 Most Significant Bits (MSBs) of the byte) is the address of the master originating the call (fig 5.21), followed by the data bytes. An intelligent device, like a microprocessor, would recognise this call and receive the data. The only device to recognise a general call in this system is the SCC68070. Although it can make a general call, this was not used as the other available devices can not recognise this call. The general call and other intelligent commands available in the bus specification are therefore not discussed in further detail.

5.3.4 I2C bus interface on the SCC68070

As described in chapter 3, the SCC68070 is provided with an I2C bus interface. It can operate in both master and slave modes and also as either transmitter or receiver. Communication between this interface and other I2C devices is controlled by a set of registers. A data register performs serial and parallel data conversions. It holds data before transmission and also after reception, until it is read by the CPU. An address register holds the address allocated to the device in the seven MSBs, and the LSB is used to program it for an 'always selected' mode [43].

An 8 bit status register reflects the most recent information about the interface (fig 5.22). The LRB (Last Received Byte, i.e. bit 0) in this register contains the information about the received acknowledge. When in transmitter mode and this bit is zero, the last transmission has been acknowledged. The next most significant bit (ADD0) indicates that a general call (00H) is detected. It is automatically reset to zero after a 'stop' condition. The 'Addressed As Slave' bit

(AAS) indicates that the interface has been addressed as a slave device and the call has been recognised. An access to the data register would reset this bit to zero. Bit 3 is used to indicate the bus arbitration status. If it is set, it indicates that the arbitration was lost during the previous attempt. Only the CPU can reset this bit. After successfully receiving or transmitting a byte or after loosing the bus arbitration, the PIN, i.e 'Peripheral Interrupt' bit (bit 4), is set to 'zero' and, if programmed in PICR1, an interrupt is generated to the CPU. It should be set to 'one' during interrupt processing, because, when this bit is zero, the interface holds SCL low and further activities on the I2C bus are blocked. A 'start' condition on the bus sets the 'Bus Busy' (BB) bit in the status register. The interface is prohibited from proceeding if this bit is set. It is reset when a 'stop' condition is detected. Bit 6, if logical 'one' indicates that the interface is a transmitter, otherwise it is operating as a receiver. Bit 7, if logical 'zero', indicates that the device is in the slave mode. A 'one' in this bit indicates that the interface is still a master.

The control register has only three flags. The SEL flag indicates that the device is selected. It is set by the 'AAS' bit in the status register and is cleared by a 'stop' or a 'repeated start' condition on the bus. The ESO flag is used by the CPU to enable or disable the interface. If it is set, the interface is enabled and when cleared it is disabled. The 'ACK' flag is used to generate the acknowledge bit on the bus. If it is 'one' reception will be acknowledged with a 'zero' bit on the SDA line. But if it is 'zero', reception will not be acknowledged. Instead the SDA line will be held high during the acknowledge period.

The clock control register holds a divisor reference number for the SCC68070 system clock to generate the proper frequency SCL signal. This register value is 'zero' at reset and is set to a non-zero value to enable the proper SCL clock.

Since the bus allows a maximum SCL frequency of 100 KHz, the clock register value of '03H' was used to generate a serial clock of approximately 96 KHz [43]. The two devices attached to the I2C bus are two PCF8574s (8 bit remote I/O expander) for the centronics port and a PCF8583, a clock calendar. These two devices are described in the following sections.

5.3.5 PCF8574-The remote I/O expander

The PCF8574 is designed as a slave device [48]. It has a bidirectional 8 bit I/O port accessible via the I2C bus and visa versa. Serial data from the I2C bus is latched at the 8 bit output port after the full byte has been received (fig 5.23). The latched output port has a high current drive capability (i.e Iout = 30 mA), therefore, no extra buffering is required. If it is used as an input port (i.e read mode), data at the parallel port is latched into an internal register before clocking it to the serial I2C port (fig 5.24). Data present at the previous acknowledge phase is latched, therefore, any data changes between acknowledgements is not captured and is lost. In the output mode, the data at the parallel port is valid after the last acknowledgment. It remains latched at the output port until a new data byte is received.

An interrupt is also available. The interrupt is generated if a logic state change occurs at the parallel port (fig 5.24). This is an open collector type output and can be used at the microprocessor to initiate a remote read or write process. In this manner these devices can be used as remote monitors.

Two devices (PCF8574) were used to provide a centronics interface capability for printer use (fig 5.25). One of these ICs (IC2) was used to provide an 8 bit parallel

data bus. The second IC (IC1) was used to provide a 'data strobe' line as an output to the printer and to monitor the 'BUSY' line as an input. If the 'BUSY' line is high, the centronics interface is not ready to receive data. When it becomes 'low', a data byte is sent to the IC2 and then the port is strobed by first writing a high to the 'strobe' bit and then a 'low' to it. The other centronics signals such as 'ACK', 'PE' (Paper End), SELECT, PRIME (initialise printer) and ERROR were not connected as all this information was available on the printer's front panel. The pin list for the centronics port is given in table 5.4.

5.3.6 PCF8583-The clock calendar chip

The PCF8583 [48] was the second device connected on the I2C bus. It is a low power clock calendar chip with '256 x 8 bit' RAM. With a minimum of external components required (physically a 32.768 KHz crystal), it offers many programmable features. Some of the features are as follows:

clock operating supply voltage 1.0 Volts to 6.0 Volts,

operating current 50 micro Amps. maximum with no I2C access,

clock function with leap year calendar,

programmable 32.768 KHz or 50 Hz time base,

256 x 8 bit RAM with automatic address incrementing,

programmable alarm, timer and interrupt functions.

The different functions of the PCF8583 are program controllable via software. The first part of the RAM (locations 00 to 07) serves as registers for clock func-

tions. These operate as parallel registers which hold the clock information (fig 5.26). Location 00 serves as a status and control register (fig 5.27).

The information held in the clock, timer or alarm registers is in 'Binary Coded Decimal' (BCD) format. The format of stored data in memory locations '04' to '07' is given in figure 5.28. The alarm function is enabled via the status register by setting bit 2 to 'one', and the alarm control is programmed in the alarm control register i.e. location 08 in the RAM (fig 5.29). The alarm operates only if the contents of the alarm registers match bit by bit the selected function counter register (i.e. clock or timer function). In case of a daily alarm (fig 5.29), only the time registers (hours, minutes and seconds) are compared. When a 'weekday' alarm is enabled, the weekday register (location '0EH' in RAM) is also used (fig 5.30). But, if the date alarm is enabled, the year and weekday are ignored. Instead only the month, date and time functions are used for the alarm function. If the interrupt is enabled (fig 5.29), the interrupt is also generated along with the alarm. Once the alarm is triggered it can only be reset by the microprocessor. This is accomplished by resetting the alarm flag (bit 1) in the status register.

The circuit diagram for the PCF8583 is given in figure 5.31. A small circuit comprising of two diodes and a resistance was added to provide a rechargable battery backup to the device. The time base was provided by a 32.768 KHz crystal.

5.4 The link adaptor interface

The transputer family normally uses the transputer links for data transfer. It is a high speed serial communication system and provides a fast and simple method

of synchronised data communication.

The link is a bidirectional, two wire data communication system. The two wires are the 'Linkin' and 'Linkout' pins on the transputer [18]. The 'Linkout' line carries the output data and is connected to the 'Linkin' pin of the next device. The receiving device, after receiving the information correctly, sends an acknowledge packet back to the transmitting device (fig 5.32). This acknowledge is sent back on the Linkout of the second device which is connected to the Linkin of the first device. This communication, i.e. link communication, is not synchronised with any external clock. Therefore it is insensitive to the clock phases and works independent of the local clock.

A link adaptor was added to the SCC68070 to provide it with the capability to communicate with the transputers used in the parallel computer system and operate as an I/O controller. The IMSC012 'Link Adaptor' was used [47]. It provided an interface between the transputer link and the microprocessor bus. The circuit diagram is given in figure 5.33. The Linkin and Linkout signals are buffered and properly terminated to avoid noise problems. Two flip-flops were used to provide the signals with the correct delays and to synchronise the data transfers between the link adaptor and the SCC68070. The required 5.0 MHz clock was provided by prescaling the 20 MHz 'clock module' output using a 74LS393.

5.5 Summary

This chapter includes an introduction to two bus systems; the SCSI bus and the I2C bus and the circuits designed around them. The SCSI bus is a local bus used

to connect the CPU to semi-intelligent independent units. It is a parallel bus and provides handshake signals to synchronise data transfer. The connected devices are usually hard disks, tape streamers and occasionally printers, plotters, laser printers etc. SASI is a subset of SCSI. A circuit was designed to provide a SASI interface to the SCC68070. A SASI interface was implemented as only disk and tape units were to be connected to this bus.

The I2C bus is a serial bus. It provides a bidirectional, two wire communication system. One line carries the serial data while the other provides a serial clock. The data on the data line is synchronised with the serial clock. The data line also allows bus arbitration while the clock line provides a handshake mechanism to allow data transfer at optimum speed. A centronics interface and a real time clock were attached to this bus.

The SCC68070 was also interfaced with a link adaptor. This was necessary as the SCC68070 Single Board Computer (SBC) was to serve as an I/O controller for a parallel computer system based on transputers. The discussion in this chapter completes the hardware description of the I/O controller board. The next chapter includes a description of the device drivers for the I/O functions working under 'Tripos'.

Chapter 6

Device drivers

6.1 Introduction

The device drivers required to implement the Input/Output (I/O) functions on the I/O board include the serial driver, the clock driver, the floppy disk driver, the SASI driver and the Inter-IC driver. The serial driver handles requests from the devices attached to the RS-232 bus. The serial bus interface circuit on the SCC68070 has two eight bit registers, one for the transmitter and one for the receiver. This interface circuit has no data buffer, therefore, the data transfers can be handled only on a character by character basis. Hence, the driver was implemented to make the transfers for individual characters. The circuit is allowed to interrupt for each transmitted or received character. If the interrupt was for a received character, the driver takes the received character to a separate data area and prepares the receiver for the next character. Similarly, if the interrupt was for a transmitted character, the driver would put the next character in the transmitter register and prepare the interface controller for the next transmit action. If more than one character is to be transmitted, the characters are added to the 'Transmitter Work Queue' (TWorkQ), and are transmitted on a one by one basis at each transmitter interrupt. Similarly, while receiving data on the

serial bus, on interrupt, the received characters are stored in a separate buffer, the Receiver Work Queue (RWorkQ), and after the buffer is full, these characters are sent to the operating system in the form of a packet. The serial driver runs with a higher interrupt priority to reduce the interrupt latency. This also helps to avoid missing characters.

Two other commands implemented in the serial driver, are the 'Get PARAMETERS' (GPARMS) and 'Set PARAMETERS' (SPARMS). The GPARMS function acquires the 'already set' parameters from the serial device and sends them back to the packet master, while the SPARMS command allows the driver to redefine the device parameters. These parameters include the serial bus speed, number of bits per character, number of stop bits and the implemented parity check. This means that the serial interface circuit can be adapted to handle a wide range of the data transfer speed and different character sizes.

The clock driver is used to update the time in the Tripos rootnode [50]. As described in chapter 3, Tripos stores the time in the rootnode in the form of 'days', 'mins' (minutes) and 'ticks'. The driver is interrupted by the timer device every 20mS. This interrupt is used to update the value of 'ticks' and consequently the values of 'mins' and 'days'. The 'delay' function is also implemented in the clock driver. The delay argument value is received as a number of ticks to be delayed. The driver, receiving the packet, waits for the required number of ticks before handing the controls back to the Tripos, and hence the delay is implemented.

Two types of disk drives were used, the floppy disk drive and the hard or winchester disk drive. The floppy disk drive was connected to the SCC68070 bus using a floppy disk controller chip (chapter 4), while the hard disk was interfaced through a SASI interface controller (chapter 5). The floppy disk driver considers

the floppy disk as a set of logical blocks spread across one or both of the disk surfaces in the form of sectors, tracks and cylinders. The information about the sector size, number of heads, sectors per track and total number of cylinders is loaded in the device 'Environment Vector' (EnVec) during the device creation process and is used by the filing system to put information on the disk surface. Once the disk is 'formatted', Tripos treats the disk as a set of contiguous blocks with the rootblock [50] in the middle. The 'read' and 'write' commands on the floppy disk drive are implemented for both the individual blocks as well as for the whole cylinder. The other commands implemented for the floppy disk driver are 'reset' (or rezero) the drive unit, 'format' and 'status' commands. The floppy disk driver is used as an example to explain the 'driver functions and implementation' latter in the chapter.

The SASI driver basically implements the read, write, reset and status commands for all SASI devices. These commands were implemented as common commands because these commands have standard features, which are the same for all the SASI devices. The reset command causes the SASI device to move its read/write heads to the 'track zero' location. In a hard disk device, connected on the SASI bus, this command would mean that the drive will seek track zero and place the head on that track. Likewise, the read and write commands were implemented to transfer the blocks of data to and from the SASI device. During the implementation of this driver, only the hard disk drive was considered to be present on the SASI bus. The format command was also added for the interfaced hard disk drive. This command, unlike the other implemented commands, is not available for all the SASI devices as this is a manufacturer specific command and was only implemented for the interfaced hard disk.

The Inter-IC (I2C) driver was used to interface with the devices connected through

the I2C bus. These devices include the printer and the real time clock (chapter 5). The I2C interface circuit available on the SCC68070 can be programmed for different speeds of the data transfer. The implemented commands include read, write, reset and, as for any serial bus, the SPARMS and the GPARMS commands. The slave device address is also received as a packet argument. This would mean that the I2C bus driver could handle a number of devices and with a number of data transfer speeds.

In Tripos, the device drivers can be written in the 'C' language. However, to comply with the existing Tripos machines, these drivers were written in MC68000 assembly language. Although all of the above mentioned device drivers are provided, the floppy disk driver is used, in this chapter, to explain, generally, the working of a driver, the device packets and the other related requirements.

6.2 Introducing 'Tripos'

Tripos has been designed as a small and portable operating system for a variety of MC68000 based computers. It is a single user-multiple task operating system [43]. Several different tasks (programs) can be loaded at a given time. However, only one task is executed at a time. Whenever an I/O call is made, a device specific routine is called. This routine would set up the hardware to perform the requested action. After this it returns back to the requesting task which can be set to wait until the I/O is completed. While this task is waiting, the operating system searches for another job to run. It continues the next program until it is requested to come back by the previous task. If it can not find a runable task, the system enters an idle loop and waits for an I/O interrupt which will then allow this task to run. Since it is a single user system, protection between the

tasks is not provided. The only protection available is through the user mode and the supervisor mode on the MC68000.

In Tripos [42] the higher level machine independent part is written in the 'BCPL' language. The lower level codes (i.e. Kernels and device drivers) are written in assembly language. The kernel consists of the task scheduler, task activation and deactivation code, code to interface the device drivers and the device libraries (DLIB and BLIB). A minimum of two tasks are present, the command line interpreter (CLI) and the debug task. Dynamic tasks can be created and used by the user.

The tasks and the devices are listed in a linked list accessible through the 'root node'. The tasks are referred to by a positive integer used as an index into the task table to point to the Task Control Blocks (TCBs). The devices are somewhat like tasks. These are referred to by negative integers and are linked through the device table to their respective Device Control Blocks (DCBs). Each TCB has a segment list which is a vector pointing to the loaded code segments. These segments are the task handlers, the device handlers and the kernels. Some of these segments (e.g. kernel) are shared by all tasks.

The data structure can be found from the rootnode. The debug task can be used to inspect and modify the data structures. When a task is created, it is in the 'dead' state. It is provided with a TCB and a segment list. It remains dormant until a packet is received. Similarly a device is in a dead state if it has no packet on its work queue. The packets are the normal means of task communication and are also used for task synchronisation. A packet is simply a small vector with different fields in it to identify the 'destination task', the 'required action' and other related information. The communication with devices is also with packets

though the 'actions' for tasks and devices may be different.

The file handlers are used to implement the 'Tripos' filing system. They provide the user with facilities to create, modify or delete files or directories. The handler translates the user requests to low level requests which are sent to the device driver as packets to read and write the disk blocks. The other standard handler is the console handler. The user interface is provided by the CLI which receives input from the console handler and executes commands.

6.3 Floppy disk driver

Floppy disk drives are well known and commonly used devices in small computer systems. A floppy disk is flexible myler disk coated with some magnetic material. The disk drive unit is equipped with a head to read and write information onto the disk surface. The information on the disk surface is stored in the form of a series of pulses, coded as logical 'zeros' and 'ones' with different orientations of the magnetic particles. When the disk is loaded, the 'spindle motor' rotates the disk with a speed of around 300 rpm, and the read/write head is allowed to touch the disk by means of some head load mechanism. This read/write head, thereafter, writes information on the disk surface in the form of a circular track. These tracks are further divided into smaller portions called sectors or blocks. The head can also move inwards (i.e away from the disk boundary), or outward towards the track 0, with a specific step size, up to an allowable number of steps, thus forming a number of tracks on the disk surface. The maximum number of tracks that can be placed on a floppy disk are limited by the disk drive as well as the specifications of the magnetic media on the disk surface. For example, an 80 track disk drive can put 80 tracks on a higher density '96 tpi' disk but a 40 track

disk drive will always put 40 tracks on the disk regardless of its higher capacity. Similarly, an 80 track disk drive unit can not put 80 tracks on a '48 tpi' disk without losing information because of the lower disk capacity. Further more, the disk drives can also have two heads (i.e. for double sided disks), thus, utilising both the disk surfaces.

Tripos, during creation of a new device interface, requires information about the floppy disk drive like the number of heads, number of tracks etc. This information is used by the file handler to place information on the disk. The lowest cylinder number represents the outer most track on the disk and usually it is referred as 'track 0'. The highest cylinder number is the inner most track and in an 80 track floppy it is, therefore, numbered as 'track 79'. Blocks or sectors are normally of 1024 bytes each. The disk device, like any other device in Tripos, can either be automatically mounted during the system 'startup' process, or they can be dynamically mounted using the 'mount' command [50]. The mount command uses the file 'Devs:mountlist'. This file contains the information related to the device which is required by the device handler. This information can be altered, or more information describing a new device can be added to this file to suit the specific device requirements.

The floppy disk driver, like any other driver running under Tripos, is expected to use interrupts. This adds speed to the system operations. Although it is possible to handle 'information transfer' by using the device polling mechanism, it might be difficult to meet the time constraints imposed by the floppy disk controller chip (chapter 4). The driver was written to use channel 2 of the DMA controller on the SCC68070. It was set up to make the block transfers, and after finishing with the data transfers an interrupt is generated by the controller chip to mark the end of the data phase. This interrupt are, then, handled and the result analysed

to check for the data transfer status.

After the device is mounted and the hardware is set ready to perform the commands implemented in the driver, the driver remains inactive until it is brought to life by a call to 'StartIO'. When the driver is invoked, there is at least one packet on the driver WorkQ. The packet is an area of memory initialised with the standard information (fig 6.1). The link field in the packet is used to link the packets together and, therefore, it holds the memory address of the next packet. The packet making an I/O request can be sent by a task either directly to the driver, or through the file handler. The packet type is the action requested for the device (i.e. write or read), and the packet arguments provide the necessary information required to implement the request. These packet are handled one at a time and the second packet is considered only when the previous packet action is completed and is sent back to the packet master.

The device drivers in Tripos consists of two parts, the device control block and the device driver code. The device control block, (section 6.3.1), is a vector, filled in partially by the operating system and partially by the user, and provides the operating system with an interface to the device driver code. The different routines in the device driver code and their functions are explained in section 6.3.2 through section 6.3.7.

6.3.1 Device control block

As described in chapter 3, the devices in Tripos work under the Device Control Block (DCB) accessible to the filing system through the pointers in the 'Device table' (Devtab) [50]. The DCB provides the file handler with the necessary

information it requires for the file manipulation (fig 6.2). Some information about the device (e.g. block size, number of heads, total number of tracks, block per track, lowest cylinder number, highest cylinder number and interleave factor etc.) is also available in the device 'Environment vector' (Envec) in the 'Device info' (Devinfo) structure of the info substructure. The device handler, at first, reads this information from the 'Envec' and makes it into absolute information related to the device and the command, (e.g read or write offsets in case of the disk device). These 'DCB and Envec', vectors are filled in with the relevant information during the device mounting process, and this information is used by both the device handler and the device driver.

The DCB structure is approximately similar for all devices other than the number of Interrupt Transfer Blocks (ITBs). The number of ITBs depends on the device type. For example, the disk device uses one interrupt and hence has only one ITB. The serial driver, on the other hand, uses two interrupts, one for each of the transmitter and the receiver. It, therefore, requires two ITBs. Two interrupts were used for the serial driver, so that the action requests can be handled with minimal interrupt latency.

6.3.2 The 'Open' routine

As the name implies, this routine is called during the mounting of the device driver. It is used to set up the initial data structure, reset the associated device and initialise the hardware to a proper ready state. The controller is also programmed with the proper parameters (i.e. step rate, head load time etc) associated with the disk drive. The FDC status register is available for reading at all times. The data register can, however, only be read or written at the time

when the device is ready for it (chapter 4). The flow chart for device opening routine is given in fig 6.3.

The Floppy Disk Controller (FDC), after reset, assumes that the drives are ready and then starts polling them to see if they changed state. If it finds that they are not ready, it assumes that the ready signal has changed state, therefore, it interrupts. In the hardware, however, the ready signal was tied low to avoid these initial interrupts (chapter 4). But, even then, sometimes, the controller gets into an unusual initial state and interrupts. Therefore, during the device opening process, four interrupt sense commands are issued to the FDC chip to remove these interrupts before programming it with the correct drive parameters.

This 'Open' routine, for the floppy disk controller, therefore, starts with acquiring the device ID and storing it in a known location. After this four interrupt sense commands are issued to the clear the interrupts from four possible disk drives. The interrupts remain off during the device opening routine, and, the status of the interrupt sense command is read out from the controller data register to make the controller ready for the next command.

After the interrupt sense commands, the 'mode' and 'specify' commands are issued to the controller [48]. The mode command sets the drive data rate, the head load and unload time and the write precompensation boundary track number. Also the implied seek mode is switched on for automatic track seeks for read and write commands. The 'specify' command sets the 'seek step' rate, motor 'on' and 'off' time and switches on the DMA mode for data transfers. In the DMA mode, the controller does not interrupt for the transfer of data bytes. Instead, it generates the DMA request signal and makes data transfers by DMA request/acknowledge handshake.

6.3.3 The 'Close' routine

The close routine, as the name implies, is used to shut down a device. A call for the 'close' routine is made by the operating system before it tries to remove the device from the device table. This call does not remove the device, but, is only used to reverse the actions previously done by the 'Open' routine. In this hardware, however, there are no specific actions required before 'closing' the floppy disk device. The only thing which can be done is to disable the interrupts from the FDC. This is done by switching off the SCC68070 interrupts in the Latched Interrupt Register (LIR) (fig 6.4).

6.3.4 The 'StartIO' routine

The 'StartIO' call is made by Tripos when it makes a request for an I/O data transfer. It is called when the first packet arrives on the WorkQ or when a previous packet has been returned by the RecallIO routine. On entering this routine, the packet action is extracted and examined (fig 6.5). The relevant action (e.g. read, write etc) is called upon to do the requested job. The action types are, read from the disk, write to the disk, format the disk, reset the disk drive, check the drive status, and switch on or off the motor. These actions are hardware dependent and are explained individually in the following lines.

A read command (fig 6.6) is used to read data from the disk surface. The packet provides the information about the disk drive unit number, the address of the buffer where data is to be put, the byte offset i.e. starting point on the disk and the size of the data to be read. The read offset was used to calculate the track or cylinder number on the disk, the head number or side of the disk, and the

sector number within the track. The read size provides the number of bytes, and hence the number of blocks, to be read from the disk surface and is also used to program the DMA controller to make the data transfers. Using this information, a command buffer is prepared for the read command, filled with the information for the FDC about where to start reading and how much to read.

Tripos normally requires a single sector for each read request. However, in certain cases where cylinder manipulation can make things faster (e.g. copying a whole disk to another disk), the full cylinder is read and transferred to the read buffer.

A write command is similar to a read command except that it writes on the disk surface (fig 6.7). The packet provides the same information, including, drive unit, byte offset, write buffer address and the transfer size. The single sector or multiple sector write is also provided including writing the full cylinder in one action.

The format command is used to reformat a floppy disk. A call to the 'StartIO' routine with the packet type as 'format', is made by the Tripos 'format' utility program. The format command is implemented to provide an International Business Machines (IBM) compatible Modified Frequency Modulation (MFM) disk formatting standard. The packet received is similar to that for a read or write command. The byte offset in the packet is used to calculate the side of the disk and the track number to be formatted. The FDC format command does not perform an automatic seek. Therefore, before formatting, a seek has to be made to the required track. As the controller can interrupt after the seek command is completed, the format command ends after issuing the seek command to the controller. Before this, the command buffer is prepared for the format command and a seek flag is set. When the controller interrupts after the seek has ended,

this seek flag is used to detect that the interrupt was for a seek command and the format command is issued to the controller to format the track (fig 6.8).

The disk is formatted on an individual track basis. During formatting, for each sector, the controller needs the track number, side number, sector number and sector code to put on the disk surface. This information is organised as format data and is transferred to FDC using the DMA controller. The track number and side or head number are the same as calculated when deciding the track to be formatted. The sector numbers are from one (i.e. the starting sector number) to five, as a single track can accommodate five sectors. The sector code is a manufacturer code for the sector size [48] which is '03' for a sector size of 1.0 Kbytes and is '02' for a sector size of 512 bytes.

The read, write and format commands are issued to the floppy disk controller through a small 'Start Command' subroutine. This subroutine causes a wait for a Test And Set (TAS) flag from the DMA channel 1 (chapter 4). The test and set semaphore was included because the faster data transfers from the hard disk on the higher priority DMA channel 1, lock out the slower but time critical data transfers from floppy disk controller on the lower priority DMA channel 2. This means that, effectively, only one data transfer is being handled at a given time despite using two DMA channels. Because mainly the hard disk is used and the floppy disks are rarely used, this does not effect the system performance very much.

After the TAS flag is cleared by the DMA channel 1, the flag is set by channel 2 to secure its turn of data transfer. This TAS flag is checked by both disk drivers and in each case it causes a wait until the other channel has finished its data transfers. The DMA controller channel 2 is, then, programmed to make data transfers and

the command is issued from the command buffer which is already prepared by the respective command subroutine. The StartIO routine then returns and, once again, the driver becomes inactive. After the data transfers have finished, the controller will interrupt the CPU and, the interrupt routine will be called to handle the interrupt.

The ResetAll command is called to restore the heads of the disk drive to the track zero (fig 6.9). The controller issues a number of step pulses and continues stepping out the head until it detects the 'track 0' signal from the disk drive. This command, therefore, requires only the drive unit number as the packet argument.

The status and motor commands require the minimum length of code. The status command, as the name implies, is used by the Tripos to check whether the disk drive is ready prior to making a read request. As mentioned earlier, the ready signal from the disk drives is grounded. Therefore, it was not appropriate to check drive status through the controller as the drive will always be in a ready state. Moreover, the controller can execute a read, write or any other command only when the drive is ready. The status command, therefore, after receiving a request, simply sets the ready status in the packet and returns it back (fig 6.10).

The motor type of action is required to switch on the motor prior to the read or write request. This is to avoid any delay which can occur while the controller is waiting for the motor to attain an optimum speed. In this hardware, however, switching the motor on and off is handled by the controller itself. Therefore, this command does nothing and simply returns the packet back to the task (fig 6.11). The command is included to provide compatibility with the operating system.

The status and the motor commands return the packets by themselves and no

call to the RecallIO routine is made. If there are more packets on the device WorkQ the StartIO routine is called again. The packet is returned to the 'master' task using the kernel call 'K-QPkt'. Before calling the kernel, the address of the returning packet is stored in the SCC68070's register D1, while, the device identity number is stored in D2. The packet is removed from the device WorkQ and is marked as 'available' (i.e. not in use). The WorkQ tail is readjusted so that the next StartIO call could find the next packet for action (fig 6.12).

6.3.5 The 'AbortIO' routine

This routine is called by Tripos to cancel an I/O request (i.e. packet), which has already been sent to the device driver. The packet in question could be, either, still waiting in the driver work queue, or the driver could have finished with the packet and sent it back to the task. A call to the standard kernel routine 'DQPkt' [50] tries to remove the packet from the driver work queue. It unhooks the packet from the driver WorkQ and removes it so that the device never knows of the packet at all. In case the packet is not found on the driver work queue, it looks for the packet in the task work queue to see if it is already back. If the packet is already back, it is, again, removed from the task work queue and is canceled. If the packet is not present on either of the work queues, then, the packet address is passed onto the driver to cancel any further action.

If a driver maintains a private work queue (e.g serial driver), it might need to abort the action and remove the packet from its private work queue. In the case of the floppy disk driver, the packets remain on the DCB WorkQ. No private queue is maintained. The AbortIO, therefore, does nothing.

6.3.6 The 'Interrupt' routine

The interrupt routine is required to respond to the interrupt from the floppy disk controller. The cause of the interrupt could be any command, i.e. reset, seek, read, write or format. In the case of the seek and reset commands, the controller is expecting the interrupt sense command to clear the interrupt. This condition (i.e. controller is in a 'not ready to read' state) can be detected by reading bit 6 and 7 in the controller status register. The seek and reset commands can further be distinguished by checking the seek flag set during the format command routine before issuing the seek command (fig 6.8). During these commands, if an error is detected, no retry is made. Instead, the Error flag is set with the type of error and a positive number is returned to mark that a packet is present and can be retrieved by a call to the RecallIO. If no error is found and the seek or reset went as expected, then, for the reset command, the RecallIO call is again requested and the packet is returned with a valid result. In case of the seek command, however, the 'Start Command' subroutine is called to perform the format command. The seek flag is cleared to avoid any later confusion and a zero is returned to mark that the interrupt has been handled successfully but no packet is present to be returned to the 'master'.

If the controller is in a 'ready to read' state condition, it corresponds to an interrupt because of a read, write or a format command. In the case of these commands, if an error is detected, a retry is attempted before handing over results back to the master. The retry flag is set and the call to the Start Command subroutine is made. If, however, the retry also fails, the disk command is abandoned. The type of error is set in the error flag and a positive number is returned to make a request that the packet must be removed from the WorkQ.

If the command was successfully completed, the error flag is cleared and a positive integer is returned to ask for a RecallIO call. In cases where the interrupt could not be handled by the interrupt routine, a negative integer is returned. This shows that the interrupt handling was unsuccessful and the system (if applicable) will try the next possible interrupt from the ITB.

The interrupt routine is kept to a minimum to reduce the delay to other interrupts. Figure 6.14 gives a flow chart for the interrupt handling process in the floppy disk driver.

6.3.7 The 'RecallIO' routine

This routine is called after the interrupt routine if the interrupt routine had notified the presence of a removable packet from the device WorkQ. The packet in question is found by looking in the DCB at the WorkQ head. The error flag is tested for any possible error. If the flag is set, the error is stored in the packet and the packet is returned to the master. If, however, no error was detected, the packet is returned with the transfer length in the 'res1' packet result field. The 'res2' packet result field is cleared as this will be looked at by the kernel and a non zero value in this area will be considered as the returned 'error code'.

Fig 6.15 gives a flow chart for the RecallIO routine. After the kernel calls the RecallIO routine, it also checks the DCB WorkQ. If more packets are present in the WorkQ, it calls the StartIO routine again and the whole process restarts. If the WorkQ was empty the StartIO is not called upon. Instead it will be called later when the first packet arrives on the WorkQ.

6.4 Starting up the system

Before the system can be cold started, at least one working version of the Tripos image file has to be ready installed on the bootable disk. This system image file consists of a number of executable binary code segments comprising the kernel libraries and the device driver codes. These segments and codes are declared in a system building command file, called the 'sysbld' file, and are copied and linked together to the Tripos system image file by the system linker 'syslink'. This system image file is loaded into the RAM during the bootup process and when executed, it places the code segments on the desired locations and fills in the rootnode and task and device tables with the primary information.

The initial system information, like the initial tasks, devices and the segments are declared in the sysbld file. Other declarations in this file include the absolute minimum and maximum locations of memory including the size of the available RAM and the position of the rootnode. At least two tasks, the Console Line Interpreter (CLI) and the debugger (debug) tasks, and two devices, the serial device and the disk device are required in the initial configuration. This initial disk device becomes the primary disk device and is always referred to by Tripos for the utility provisions. The serial device is required because, without it, the user will not be able to communicate with the system.

After creating the Tripos system image file, it is installed on the disk, so that it can be loaded into the RAM during 'bootup'. The 'Install' utility program was provided to put installing information on the disk. Although the image files can be installed using the 'disk editor' (disked) [50], it is easier and safer to use the install utility.

During executing the Tripos image file, the secondary startup information is provided in the 's/startup-sequence' file. This file is a command file and the user can alter the contents in here for a desired initial setup.

6.4.1 Installing 'Tripos'

The Install utility, when invoked, reads the 'key' to the file to be installed and writes it in sector 1 track 0 of the disk (also referred to as block 0.) at the desired location. A typical command line would be

install device:file-name version a ¡CR¿.

The file name would also include the logical device name to be installed. The version is optional, however, if it is not mentioned in the command line, it will default the installation as version 'A'. All the versions from A through Z are possible and can be installed on a single disk.

After reading the device name in the command line, the 'info' substructure in the rootnode is examined to find the required device. To do this, each device listed in the substructure is accessed and the name is compared. If it is the same as the required device, the device ID is read from the data structure. If, however, the device is not already mounted (i.e. ID = 0), then, the device driver's object code is loaded and the device is created by filling in the necessary information in the info substructure. Eventually, the device ID is acquired and the device is linked to the device table at its ID position.

The key to the file is acquired by creating a filing system lock on the file. This lock is obtained by a call to the 'locateobject' kernel routine. The key is actually the disk block number of the file header block for the Tripos image file. This block number is placed on block 0 of the disk and is use in the bootstrap. To avoid overwriting the other pre-installed image files, the disk block 0 is first read into the RAM and the key is copied to the desired version location prior to writing the block on the disk. Block 0 of the disk is reserved for installing the system image files and remains hidden to the filing system. The flow chart for the install utility is given in figure 6.16.

6.4.2 The bootstrap

The flow chart for bootstrap is given in figure 6.17. It is a ROM (Read Only Memory) resident code and is invoked during the 'cold start' of the system. The bootstrap works as follows.

In the beginning, after the reset signal is removed, the SCC68070's Program Counter (PC) is loaded with the ROM address from the reset vector location. The ROM code, therefore, starts executing. The VSC registers are initialised to provide a proper RAM refresh and read timing. The stack pointer and the store pointers are loaded with initial values and different RAM locations (i.e. pointers and variables) are filled in with their initial values. After this, the serial interface is brought to a ready condition to read 'device' and the desired Tripos 'version' from the console.

Once the device and version are read in from the console, the disk device is initialised. This device initialisation is the same as the previously mentioned

device opening routine (section 6.3.2) except that there is no device ID involved. For example, if it is a floppy disk device, then, four interrupt sense commands, the mode and the specify commands are issued and the disk drive is rezeroed. Since the bootstrap is working in the polling mode and the DMA controller is not used, therefore, it is not initialised. Interrupts remain off during the bootstrap process, therefore, the device can be polled for reading data without any interruption.

After initialising the device, the disk block 0 is read to get the 'key' to the desired image file. This key is copied over from the desired version location, and from this block onward, the next block field in the disk data block is used to get the key to the next data block. When a block is read, it is checked for a checksum error [50]. If an error is found, an endless number of retries are made to read the block correctly.

When a block is correctly read, the data from the read buffer is copied over into the store and the store pointer is incremented. This process of reading blocks and coping the data into the store continues until the end of file. After reading the full system image file, the program counter is loaded with the initial stored value and the Tripos operating system starts executing.

6.5 The calendar clock

The clock calendar chip 'PCF8583' is connected to the system on the I2C bus (chapter 5) [56]. The I2C driver is used to make data transfers to the clock chip. As this bus can have more than one device at any time, the driver is organised to address any of the devices on the bus. The packet received by the driver has, therefore, the slave chip address as a packet argument. The format of a packet

to the I2C driver is given in fig 6.18.

6.5.1 Setting time

This utility was provided to set and update the time in the clock chip, and to initialise the chip for proper clock function. When the system is started, and the clock chip has not been initialised previously, some random data could be present in the chip memory. This data, if read and stored in the rootnode as the 'present' date, could upset the system because Tripos keeps on accessing the time in the rootnode every time it updates a file on the disk or displays a file or a directory. The set time program, when used, reinitialises the chip with the proper initial values (i.e. day, date, time etc), after which, the calendar is updated by the chip itself using the on-chip data base. Once initialised, the chip can remain in the proper mode with the battery backup (chapter 5) providing enough standby current. The set time program can also be used to update the clock chip RAM contents whenever a change in time (i.e. summer and winter time) occurs.

To set time in the clock chip, initial values of days, mins and ticks are acquired from the Tripos rootnode. It is, therefore, necessary to correct the date and time in the rootnode before setting this in the clock chip. The Tripos utility program 'date' was used to correct the time and date in the rootnode. The Tripos kernel routine 'rootstruct()' [50] is used to point to the rootnode. This time from the rootnode is stored in the on-chip RAM and the clock is initialised to start the time counter. While writing into the RAM, the chip's clock function is stopped, to avoid a prestart count, and is started again after finishing the writing. Fig 6.19 gives a flow chart for set-time program.

6.5.2 Restoring time

The restoring time utility was provided to retrieve the most recent time and date information from the clock chip. The days, mins and ticks stored in the on-chip RAM, and the elapsed days and time, after storing the previous days, mins and ticks, are retrieved and added together to make the present date and time. This present date and time is stored in the rootnode and, hence, a time update is provided.

When the computer is switched on, this utility is used to initialise the correct time and date in the rootnode. This saves the effort involved in correcting the date and time every time the system is switched on. Once the date and time is corrected in the rootnode, Tripos keeps it correct with the help of the 'timer' device (chapter 3), as long as the system remains powered up. This utility was put in the 'startup-sequence' file. This file provides the computer with the information about starting up the system. Unlike the installed 'Tripos' file which is loaded and run under the 'bootstrap' (section 6.3), the startup-sequence file is an editable command file and was modified to include the restore-time utility. This provides an automatic time update when the system is switched on. Figure 6.20 gives a flow chart for restoring time.

6.6 Summary

An insight to the device drivers working under Tripos is given. Tripos was used as the operating system for the hard disk and input/output controller board. The floppy disk driver was chosen as an example to explain this. The two parts of the driver have been discussed. These are the DCB and the driver code. Different

routines in the driver code have been individually described.

In a latter section, starting up of the system has been described including the bootstrap and the methods of installing a Tripos image file. The workings of the Install utility was explained. At the end, the setting of time through the clock calendar chip has been explained. Two utilities regarding the clock chip were included in the discussion. These are used for setting the time in the clock chip's on-chip RAM and for subsequent retrieval of the up-to-date time.

Chapter 7

Coprocessor interface

7.1 Introduction

The previous chapters of the thesis described the design and functional implementation of the Input/Output (I/O) board. This chapter and the one after this is dedicated to describing the design of the graphics board and the organisation of a possible display file in the display memory. This display memory (or display RAM) was intended to be updated by the T800 transputer working as a coprocessor to the VSC, while the pixels are displayed onto the graphics screen by the VSC. The T800 which formed a processing node with its independent memory, the T800 coprocessor interface to VSC, and the colour palette interface circuit are described in this chapter. The VSC master/slave operations and the display file organisation for Screen Level Parallel Graphics (SLPG) are described in the next chapter.

The T800 [18] is an INMOS 32 bit processor with an integrated 64 bit floating point unit. It uses a 32 bit multiplexed address and data bus which provides the T800 with a linear addressing range of 4 Gbytes. Five general purpose strobes are provided which can be configured to suit a particular memory device. In

addition, a 'Memory Wait' (MEMWT) input signal is available to extend the memory cycles for slower memory devices or a data exchange to the memory mapped I/O devices. This 'MEMWT' input was used to extend the memory cycle for the coprocessor interface.

The T800 also supports Dynamic RAM (DRAM) refresh. These refresh cycles are user configurable or can be selected from a previously defined table during the system bootup process. Appropriate configuration was selected to meet the requirements of the memory devices used for the T800 independent memory module. These memory devices were 1 Mbit (256 Kbytes x 4bit) DRAM devices with the address inputs multiplexed for rows and columns.

Two groups of memory devices were made accessible to T800 excluding the on-chip 4 Kbytes of T800 'static RAM'. These groups consist of 1.0 Mbytes of local memory for the T800 processing node, and 0.5 Mbytes of display memory. The local memory was provided to run the 'Helios' operating system. It is refreshed and maintained by the T800. The video display memory is refreshed by the VSC. The T800 reads and writes to this memory via a handshake on the VSC's coprocessor interface which is described later. Pixel information relating to the picture to be displayed, is updated by the T800 in this display memory. To do this, the T800 access to the display memory is decoded and the coprocessor interface is activated. The T800 memory cycle is then delayed by asserting 'MEMWT' until the data transfer can take place.

The VSC [17] is a stand-alone VLSI graphics processing chip which generates high resolution pixel information from the display memory along with the required synchronisation signals to drive a video monitor. It can support both, interlaced and non-interlaced pictures. In addition, it can also act as a system controller for

68000-based computers, providing an interface to ROM, multiplexing the address for DRAM devices, and generating other signals, e.g. Reset, Data Acknowledge and Bus Error, for the CPU. The VSC's graphics facilities include 4 or 8 bit pixel coding for up to a 768 x 560 pixel picture, bitmap or compacted display file manipulation, and dynamic image control thus enabling the use of subscreens. It is also equipped with a pixel accelerator to provide bit and block manipulation.

The VSC is provided with a coprocessor interface for an intended coprocessor MN83502, (recently available from Philips), to facilitate generating lines, filling areas etc. A T800 transputer was used to replace this coprocessor and to provide fast communication with the existing transputer based computer. The block diagram of the graphics board is given in figure 7.1.

The VSC memory cycle is divided into two time-slot windows. The first window is used to fetch pixel information from the display memory for the current pixel and put it on the pixel bus. DRAM refresh is also provided in this window during the line or field retrace periods. The second window is available for the CPU (currently SCC68070) or a coprocessor for possible access to the memory. The VSC arbitrates between the CPU and the coprocessor. Handshake signals are produced by the VSC to grant a DRAM access request to the coprocessor to synchronise the address and control information with the data transfers. This handshake protocol defines the functions of the coprocessor interface circuit for the T800 when it is accessing the display memory.

The VSC chips can further be configured as master or slave devices. A slave VSC does not generate the video synchronisation signals. Instead, these signals are input from a master device and in response the slave device generates the corresponding pixel output for the video display. This allows the extension of

pixel size from 8 bit to 16 bits, thus giving a greater colour depth.

Dynamic image control is provided through a set of VSC control registers and two picture control areas. These are called the Image Control Area (ICA) and the Dynamic Control Area (DCA). The picture control information is placed in the display memory along with the picture information. Using this control information, the VSC can update its display control registers (i.e. display command registers, video start register etc). This updating of registers provides the potential for parallel graphics (SLPG).

7.2 T800 processing node

The INMOS transputer is considered to be an important development which has had quite an impact in the field of multiprocessor systems. The T800 chip is a 32 bit RISC microprocessor with on-chip floating point unit. It is also provided with an on-chip 4 Kbytes of RAM. With four, DMA controlled, high-speed links, it is designed as a building block of a parallel processor system. The on-chip 64 bit floating point unit operates concurrently with the CPU and can achieve sustained floating point performance of 1.1 Mflops (for 64 bits operations) at a 20 MHz processor clock frequency [18].

The T800 has a 32 bit wide memory interface. It uses multiplexed address and data signals on the same 32 bit wide memory bus. While using the 32 bit data bus, a data rate of 26.6 Mbytes/second (at 20 MHz) is possible. The on-chip (4Kbytes) RAM provides a single cycle memory access and can give a data rate of up to 80 Mbytes/second. A built in memory controller provides DRAM control and refresh timing. The memory controller also supports memory mapped peripherals. These

peripherals and the memory data transfers are supported by the on-chip DMA.

A DMA block transfer mechanism is used to transfer data and other information to the other transputers using high-speed links. Processing continues while the data is being transferred on the links. The data transfer rate depends on the link speed which is independent of the processor speed.

The T800 uses a 5 MHz input clock to generate the internal (20 MHz) clock to synchronise its internal functions. The internal processor clock speed is selectable though the PSS0-PSS2 pins. This was selected as 20 MHz for the initial experimental purposes on a wire wrap board.

The transputer can bootstrap from a ROM. The bootstrap ROM is placed at address '7FFFFFFEH'. During this the BFR pin should be high. For the system constructed, however, this pin was tied low and the bootstrap information was provided on one of the four available links. The T800 was booted with the 'Helios' operating system in the same way as the other transputers in the parallel processing system.

During refresh cycles, the on-chip DRAM controller sets part of the address bus (AD2-11) as the refresh address and generates the required strobes. The refresh controller is incremented after every refresh cycle and depending upon the selected memory configuration it refreshes the DRAM after every 18, 36, 54 or 72 clock periods of the (5MHz) input clock. The 10 bit refresh address means that '1M x n bit' DRAM chips can be used. This DRAM controller also provides the timing for the memory interface. One of the thirteen predefined internal configurations can be selected by connecting the MEMCNFG pin to one of the D0, D1 or AD2-31 pins. Alternatively, the MEMCNFG pin can be connected

to one of the ADn pins through an inverter and the RAM configuration can be provided by an external ROM. This MEMCNFG pin is sampled only at the reset before starting the DRAM refresh.

Each processor memory cycle is divided into six 'T' states (T1-T6). These 'T' states extend from half to two internal clock cycles or 1 to 4 of 'Tm' periods. 'Tm' is half of the internal clock period (i.e. Tm = 25 nS at 20 MHz internal clock). These 'T' states provide the control signals (MS0-4) for the memory interface [18]. Separate write strobes (MWB0-3) are provided to write the individual bytes into the memory during the write cycles. A read strobe, MRD, is available for a 32 bit wide read cycle. The 'T' states, T1-T3 and T5-T6, have preprogrammed durations defined during the memory configuration. T4 can, however, be extended by using MEMWT to suit the external hardware requirements.

7.2.1 Memory map configuration

As described earlier, the T800 has a linear addressing range of 4 Gbytes. This full address range will rarely be used for the RAM or I/O devices. Moreover, the data transfer between the transputers was intended to occur via links, i.e. no shared memory was to be used. Therefore it was not considered necessary to fully decode the complete 4 Gbytes address range.

The address decode circuit latches the upper 8 address bits from the T800. This divides the T800 addressing range into 256 pages of 16 Mbytes each. This was considered as sufficient allowance for future expansion. T800 local memory was placed in page '80H'. The T800 on-chip 4 Kbyte RAM also resides in this memory space. This meant an overlapping of 4 Kbytes between the two memories (i.e.

between on-chip RAM and the interfaced local RAM). This does not produce any undesired effects. However if desired, the external overlapped RAM can be accessed within the 16 Mbyte page by using the address format '8000xnnnH' (x = 0 for on-chip RAMs and non-zero for external DRAM).

The VSC controlled display RAM was placed in page '00H'. A 20 bit address word in the VSC page provides access to the full 2 Mbyte VSC controlled RAM, ROM and I/O areas. The colour palette registers and colour RAM were made available in page '01H'. The full memory map for the T800 is given in figure 7.3.

7.2.2 Memory interface

The external 1.0 Mbytes of DRAM which was added to the T800 processing node as its local memory were 1 Mbit high density DRAM devices arranged in 256K x 4 bit words. These memory devices use nine multiplexed address lines (MA0-MA8) for rows and columns.

The T800 data bus is 32 bits wide, therefore the lower two bits on the T800 Address/Data bus are not used for memory addresses. For write or refresh memory cycles, the upper 30 address bits (AD2-AD32) with the required memory address are first placed on the bus. This address is latched in the external hardware. The lower two bits, D0 and D1 (fig 7.2) are also asserted along with the address to indicate whether it is a write cycle or a refresh cycle. D0 indicates that it is a write cycle while D1 indicates the refresh cycle. These were also latched along with the address. The second phase of the memory cycle produces a memory refresh or a data transfer. All the 32 bits of the Address/Data bus are used in data transfers.

The memory was organized in four byte words. In the read cycle, the T800 asserts the MRD signal. This is an active low signal and tells the external hardware that the cycle is a 'read' cycle. All the 32 bits are fetched and are captured at the end of the cycle. For a 'write' cycle, four strobes (MWB0-3) are available to identify the individual bytes involved. One or more MWBn strobes are asserted to perform the required size of the write operation.

The external memory cycle 'T' states (T1-T6) further generate five general purpose strobes (MS0-4) available on the output pins. These strobes can be used to synchronise the external activities of the processor. The MS0 signal is an 'address valid' strobe. This was used to latch the address into two registers U7 and U12 (fig 7.2). The inverted MS2 signal was used to enable the output of the address buffer U2. Since MS2 is not driven in the beginning of the cycle (fig 7.4), the output of U2 was enabled while the output of the address latch U7 was disabled. The MS1 strobe was used to provide the decoded Row Address Strobe (RASN) through the PLD U13 [34]. The PLD listings are given in Appendix D. These listings were produced, compiled and the PLD device was programmed under 'AMAZE', the Signetic's software package for PLD programming [51].

When the MS2 signal becomes asserted, the address buffer was disabled and the output of the latch U7 is enabled. After this the T800 asserts the MS3 strobe. This was used to provide the decoded Column Address Strobe (CASN). Then the data cycle begins. The write strobes (MWB0-MWB3) selectively enable the write inputs of the pairs of the DRAMs constituting the individual bytes thus enabling the writing of separate bytes into the memory. The read strobe, MRD, enabled the output of all the memory chips, giving a possible 32 bit data read.

The memory configuration pin, MEMCNFG, was connected to the AD6 pin to

choose the AD6 internal memory configuration. For this configuration, the external bus cycle consists of five processor clock cycles (i.e. a total of 250 nS). The timing diagram for this configuration is given in figure 7.4. This provided the most appropriate timing for the TC514256 [52] memory devices used in this circuit. A faster time, by connecting the AD5 pin to MEMCNFG, did not provide the required RASN 'precharge Time' (Tpr). The minimum value of Tpr for the memory devices was 80 nS but, the memory configuration using AD5 only provided 75 nS and was, therefore, not used.

The T800 DRAM 'refresh cycles' are periodically distributed among the memory cycles. In the AD6 configuration, these refresh cycles are performed every 72 input (5 MHz) clock periods. The TC514256 memory devices have 512 rows and needed refreshing every 8 mS. For this number of rows with the selected configuration, the memory devices are refreshed every (512 x 72 / 5 MHz =) 7.37 mS. This falls within the device limits thus performing the memory refresh successfully.

7.2.3 Link protocol

A transputer has four links which enable it to connect to up to four other transputers in parallel. These links are normally used for inter-transputer communication in parallel computers. These are bi-directional data lines and operate at a speed which can be selected as 5, 10 or 20 Mbits per second using the LS, L0S and LnS input pins (fig 7.2). The data on the links is transmitted as individual bytes framed with 'start' and 'stop', bits. The receiving transputer acknowledges every received byte.

The above link speeds are available with the standard 5MHz transputer input clock. The 20 Mbits/sec speed may be reduced if the memory devices can not operate at this speed of data transfer or if all the four links are operating. Link speed can also be reduced if some of the memory bandwidth is occupied by the processor. The link signals are low in their inactive state. A high signal causes the start of the data transmission. The transputer on the receiving end acknowledges the reception while the transmission is still in progress, thus avoiding a bit slot for the 'acknowledge bit'. These links were buffered (fig 7.2) and appear active when they are not connected. Therefore, to avoid error conditions, provisions were made to ground any unused link inputs (fig 7.2).

During the bootup process, after the 'reset' signal is removed, the T800 waits for a bootstrap message to appear on one of its link inputs. The first byte after reset is taken as a control byte. This byte, if greater than '1', indicates the number of bytes that are to follow in the bootstrap message. This number of bytes is read from the link and copied into the T800 local (4 Kbyte) memory. After the 'byte count' is over, the program execution begins from the starting address.

If the control byte is '0' or '1', the transputer performs 'poke' (i.e. write) or 'peek' (i.e. read) memory operations respectively. In the case of a poke command, the four bytes following the control byte are taken as the 'write address' for the memory while the 4 bytes after these are taken as the 'write data'. For a poke command, therefore, the transputer inputs a total of nine bytes including the control byte. For reading a desired memory location via a link (i.e. a peek command), after transmitting the control byte as '1', four bytes are transferred down the link as the 'read address'. The transputer reads the required memory location and transmits the data back to the requesting device on the same link. The poke and peek commands act on 32 bit data size only. These commands

were used to test the T800 local memory.

After the memory tests proved satisfactory through the link, the T800 processing node was booted with the 'Helios' operating system [53]. Then more test were performed to check the operations of the processor and the links working together. The testing involved writing random data into randomly chosen memory locations and then reading it back to verify whether the write operation was successful. The data read was sent back from the transputer via a link to the I/O processor. In this way, the link-I/O protocol between the transputer and the I/O processor was checked.

7.2.4 Circuit description

The circuit diagram for the T800 processing node on the graphics board, along with its local memory, is given in figure 7.2. The local memory interface circuit works as follows.

When the T800 is not accessing the memory, all 'MSn' strobes (i.e. MS0-MS4) are high. When it starts accessing the memory, it places the destination location address on the memory 'Address/Data' bus and asserts the 'MS0' signal. The level triggered latch, AM29841 (U12) is used to latch the address for subsequent decode. The latch operates with the MS0 signal so that after allowing the propagation delays the decoded signals are available by the time MS0 becomes active. The lower 18 bits of the 30 bit address word (i.e. AD2-AD19) are used to address the memory contents. The upper half of this 18 bit address word is clocked into U7 i.e. AM29823, with the falling edge of the MS0 signal, for decoding the memory group and the other signals (fig 7.2).

The local memory access was decoded by U13, the PLS153 (Appendix D). The Row Address Strobe (RASN) signal was asserted by the MS1 strobe. Meanwhile, because the MS2 signal was not active, the buffer U2, i.e. AM29827, remained open and hence the row address was presented successfully to the memory address bus. When the MS2 strobe became asserted, the U2 output was disabled and the U7 output became enabled. At this stage the T800 asserted the MS3 signal. This signal was used for decoding the Column Address Strobe (CASN). Because the column address was already present from the U7, no delay was needed for the address set up. In this way, the row and column address multiplexing was performed within the specified timing limits of the DRAM devices.

For a read memory cycle the MRD output signal from the T800 was used to enable the DRAM outputs. This signal enabled the output of all memory devices and data became available on the 32 bit memory data bus. The T800, after acquiring the data, removed the strobe signals to end the memory cycle. This removed the latched address in U12 and hence the decode signals were removed. For a write cycle, MWB0-MWB3 were used to 'write enable' four pairs of DRAM devices comprising individual bytes. This allowed the selection of one to four bytes.

Memory refresh cycles also generated the 'MSn' strobes. But because the MRD and MWBn signals were not asserted, these refresh cycles did not produce any data on the memory data bus. During this cycle the MRF signal was asserted by the T800. This was used to decode a RASN signal exclusively for the local DRAM so that the refresh could be performed for the local memory only.

The local memory interface worked with the selected memory configuration and required no wait states. Wait states were only required for access to the display

DRAM. The implementation of these wait states, to extend the T800 memory cycle for display memory access, is described in the later section.

7.3 T800 coprocessor interface to VSC

The coprocessor interface is a SEQUENTIAL logic circuit which allows the SCC68070 and the T800 to share the display memory. The two accesses are serialised by the VSC. This interface also provides a means of indirect communication between the two processors and allows the T800 to generate and process the image information in the display memory.

The VSC directly interfaces the SCC68070 (I/O processor) memory bus to the DRAM bus. The SCC68070 has a 24 bit address bus out of which the VSC decodes 21 bits providing a memory map comprising the DRAM, ROM and I/O area (Appendix A). The VSC DRAM interface provides the multiplexed address for the memory devices along with the row and column address decode signals. This DRAM combines the SCC68070 system memory and the video or display memory. There is no defined boundary between the system and the display memory. However, because the first memory bank is active, the VSC uses 20 bits as the display start address. The picture information, therefore, must be placed in the first Mbyte of memory. The VSC arbitrates between the CPU, the video controller and the coprocessor. It also generates the Data Acknowledge (DTACKN) for the CPU when the data has been made available on the 16 bit system data bus.

The two processors involved (i.e. SCC68070 and T800) operate asynchronously with separate clocks. The VSC also has a different clock. The VSC synchronised

memory cycles for both processors, therefore the coprocessor interface had to be synchronised using the VSC clock. The interface circuit was implemented using the 15 MHz 'XT/2' clock output from the VSC. A faster synchronisation using the 30 MHz VSC clock introduced noise and gate delay problems.

When the transputer attempted to make an access to the display memory, it was decoded and the T800 was forced to a wait state using the MEMWT signal. The memory address and the control information was latched. The control information included the read/write address and the data size (fig 7.5). The coprocessor memory access request was made to the VSC by asserting Cycle Request (CYREQN). When the cycle request was acknowledged from the VSC (i.e. CYACKN was asserted), the latched information was placed on the video memory address (VA) and data (VD) buses (fig 7.2). After the VA and VD information had been acquired by the VSC, these buses were released and the rest of the cycle was allowed to proceed. For the write cycle the data was placed into the data latches before it was presented on the memory VD bus. During the read cycle, the VD bus was released and the data from the VSC memory bus was latched for the transfer to T800.

7.3.1 VSC memory cycle

The VSC memory cycle consists of 16 (30 MHz) VSC clock periods [17]. During the display window, the VSC can either use fast page mode to fetch two 16 bit words (using 9 clock periods), or it can use normal mode and fetch one 16 bit word (using 8 clock periods). These modes can be selected and enabled through the VSC Control and Status Register (CSR). The DRAM is also refreshed in the display window during the display inactive periods (i.e. line and field retrace

periods). When the display is inactive and the DRAM refresh is over, this window is used to access ICA and DCA information. If the ICA and DCA are also inactive, then, this window terminates.

The second window (i.e. CPU window) is only activated when the CPU or coprocessor makes a request for memory access. The VSC uses the coprocessor interface handshake signals, Cycle Request (CYREQN) and Cycle Acknowledge (CYACKN), to arbitrate for the coprocessor access. The timing diagram for the handshake signals and their relationship with the VSC clock is given in figure 7.6. The coprocessor interface was required to exchange data between the display memory and coprocessor, and between VSC registers and the coprocessor. An access to the VSC registers allowed the transputer to reconfigure the VSC by rewriting the control information into the VSC registers.

For the coprocessor interface, the T800 clocked at 20 MHz and was synchronised to the VSC operating at 30 MHz. The two clocks were independent and had no phase relation. Moreover, the VSC memory cycle was fixed. Therefore, the T800 memory cycle had to be extended to incorporate VSC data transfers. The corresponding wait states were generated for the T800. The 'wait state' time for a 'write' cycle was less than that for a 'read' cycle. This was because the write cycle on T800 part ended after the data had been latched into the data buffers, while for a read cycle, the T800 had to wait until the data had been made available on the VD bus and subsequently latched.

7.3.2 Coprocessor cycle interlock

As described earlier, the T800 memory cycle is divided into 'T' states (T1-T6). These 'T' states define the internal state of the processor and generate external interface strobes to interlock with the external devices. The T800 memory cycle can be extended by using the MEMWT input. This is an active high input signal and so to terminate the memory cycle it has to be forced low. The T800 samples the MEMWT signal close to the end of its T4 state during the high period of the processor clock output (CLKOUT) signal. If the MEMWT signal is sampled high, the T800 extends its memory cycle and stays in T4 state until it is sampled low. Only then does the T5 state start and the data transfer takes place.

The coprocessor (T800) was directly connected to the display memory bus and transferred data without involving the CPU (SCC68070). It was given a similar access to the display RAM as the CPU. A 20 bit memory address word along with some information about the data size i.e. equivalent to Upper and Lower Data Strobes (UDSN and LDSN for SCC68070) was provided on the combined VA and VD buses during the information transfer phase. If the coprocessor access was granted, this information transfer phase occurred just before the end of the display window. The VSC asserted the CYACKN signal (active low) and the 'data transfer information' (info) was made available on the VA and VD buses by the rising edge of the CYACKN signal. This was achieved by enabling the data latches (U14, U16 and U20) with the decoded INFON signal.

7.3.3 Circuit description

Initially the circuit was designed to interface the lower half of the 32 bit transputer bus to the VSC controlled display memory data bus. The picture was organised in 8 bit per pixel mode and the VSC was set for that mode. This reduced the picture resolution because in this mode the horizontal picture resolution is approximately half that of the 4 bit per pixel mode. It was, therefore, decided to modify the circuit for the higher resolution.

The complete circuit diagram for the graphics board with the above mentioned modification is given in figure 7.2. For a coprocessor cycle the display memory address along with the data size was acquired by the VSC near the end of the display window after the CYACKN signal had gone high (fig 7.6). During the coprocessor window, the VSC accessed the destination memory location and the data transfer took place during the second CYACKN low period. For the 'write' cycle, the data had to be made available before the second CYACKN occurred. Similarly, for the read cycle, the data was available by the second CYACKN period. The memory bus was required to be released before the end of coprocessor window for a proper start of the display window.

The circuit function started with the T800 performing a memory cycle for display memory. It placed the address on its ADn bus. This address was latched and decoded as described in section 7.2.1. The decoded display memory access (VSCMEMN) along with the MS0 strobe generated the memory wait state. This wait state halted the T800 memory cycle in its T4 state (fig 7.7).

The (inverted) VSC clock output signal, 'XT/2' clocked in the decoded display memory access (VSCMEMN) into the 74F273 register (U26). This latched signal

was used for generating the CYREQN signal through the PLDs U27 and U28 (Appendix D). The inverted clock was used to make sure that the CYREQN signal was present at the VSC input before the falling edge of 'XT/2'. The MS0 strobe also latched the memory address available on the T800 bus into registers U14 and U16 (2 x AM29821). The imposed wait state from U13 continued until the cycle was acknowledged. However, during the write cycle, the write data along with the decoded signals 'WHN and WLN' were also latched. These decoded signals represented the upper and lower bytes in the display memory.

When the cycle acknowledge occurred (i.e. CYACKN signal strobes low), it was latched. One XT/2 cycle later, when the row address Strobe (RASN) was sampled high, the INFON signal became activated, which enabled the address and information latches U14, U16 and U20.

The falling edge of VRASN was used to remove the INFON signal. This in turn disabled the 'info latches' which released the VA and VD buses. This VRASN signal also removed the CYREQN signal. If the T800 was performing the write cycle, the data latches were enabled and the write data became available on the VD bus. For a read cycle, the read data was latched by the rising edge of the VRASN signal (i.e. near the end of memory cycle). The timing diagrams for the coprocessor read and write cycles are given in figure 7.8 and figure 7.9 respectively. This VRASN edge was also used to end the T800 memory cycle. The wait state was removed and T800 started the 'T5' state. After the data had been acquired, the T800 removed the external strobes and terminated the bus cycle as normal.

The circuit was built so that a higher pixel depth could be obtained by using two graphics boards in parallel. Two VSCs when working together would produce

8 bits per pixel (i.e. each contributing 4 bits). The two transputers working on the two graphics boards produce similar display files. When transferring this data to the display memories, the picture data in the transputer memory (i.e. 32 bit transputer words in the second T800) would be shifted by four bits to the desired bit locations before transferring it to the (2nd) VSC controlled display memory (fig 7.10). The first T800 does not need any data shifting. In this way, the two VSCs produce the desired 8 bits of information (4 bits each in parallel) contributing the right nibble. The display file remains organised as for the 8 bit per pixel mode. The data transfers on the T800 are 32 bit words while the decoded WHN and WLN signals provide the corrected nibble transfer on the VSC end.

7.4 The colour palette interface

The two VSCs in parallel (each contributing 4 bits) generate an 8 bit pixel output for the video monitor. One way of producing the 'Red, Green and Blue' (RGB) output from this 8 bit pixel output could be by decoding colour values directly from the output by either three different 'Digital to Analog' D/A converters or a simple resistor ladder network (fig 7.11). This simple colour-component encoding scheme has the disadvantage of limiting the range of available colours. A more flexible scheme is to include a colour look-up table, or colour palette. The 8 bit pixel output values from the VSC can be treated as addresses pointing into a table of colours defined as the individual red, green and blue components in the 'colour RAM'.

The BT471, the colour palette used in this circuit (U34 in fig 7.2) had an 8 bit input address [54]. This address input generates three 6 bit colour outputs from

the colour lookup RAM feeding to the on-chip video A/D converters. The timing diagram for the colour palette interface is in given in figure 7.12. The BT471 was interfaced to the T800 without involving the SCC68070. This was done because the colour palette RAM can be updated by the T800 much faster than by the SCC68070. The register select inputs signals (RS0-RS1) specify which part of the colour palette is to be accessed. The T800 address lines AD2-AD3 were connected to these (RSn) input signals whilst the 8 palette data pins were connected to the lower byte of the T800 data bus. This implies that the register locations were separated by an offset of 4 bytes each. The address from the multiplexed Address/Data bus of the T800 was latched and decoded for the colour palette access as described in section 7.2.1. The address was latched because it had to remain valid throughout the BT471 bus cycle. If the access was decoded as 'valid' (i.e PLTSELN became asserted through U13), then the PRDN and PWRN signals, which were decoded through the PAL16R8 (i.e U33) [33] became asserted (fig 7.16). When these signals appeared, the address was already valid (on A1-A2) and the bus cycle continued as normal. At the end of the read cycle, the T800 captured the data and removed the MS0 signal. This removed the latched address and hence the PWRN and PRDN signal were disabled. This terminated the BT471 bus cycle. For write cycles, the data on the BT471 was latched by the rising edge of the PWRN signal, at the end of the T800 bus cycle.

The RGB outputs of the BT471 were synchronised with the CSYNCN signal from the VSC. These RGB outputs could directly drive the 75 ohms inputs of a video monitor. A voltage reference for the colour palette on-chip D/A converters was provided by two diodes (D1 and D2) and a transistor (Q1) (fig 7.2). The clock input to synchronise the BT471's internal activities was provided by the 'VSC pixel clock' output (PCLK). This synchronised the two pixel generating chips (i.e. the VSC and the colour palette).

After switching on the system, the colour palette RAM had to be written to by the processor. To do this, the processor wrote the address into the colour table of the base of the block to be updated into the 'RAM write address' register. Then three consecutive bytes of colour data (one byte for each Red, Green and Blue) were written into the 'colour palette data register'. After each set of three byte writes, the 'RAM address register' was incremented automatically and the 'colour palette data register' became ready to receive the colour arrangement for the next location. Similarly for reading the colour palette RAM contents, the processor first wrote the specified address into the 'RAM read address' register. The three colour bytes could then be read from the colour palette data register in the same order as above (i.e. Red, Green and Blue). The address register was again incremented automatically and the next colour arrangement was read directly without rewriting the 'RAM read address' register.

The BT471 is also provided with colour overlay registers to facilitate a cursor overlay. The overlay colour data was read and written by first writing to the address registers and then, accessing the overlay data register for the colour information again in the same order as for the colour RAM (i.e for Red Green and Blue). This facility was not used. Instead the cursor overlay was provided by writing directly into the display file.

The pixel select inputs, P0-P7, provided the address for pixels in the colour palette RAM. These inputs were latched by the BT471 on the rising edge of the internal clock (PCLK). The synchronising and blanking signal inputs (SYNCN and BLANKN) from the master VSC were also latched to maintain synchronisation. The colour palette was put on the master graphics processing board because it required the CSYNCN signal from the master VSC for synchronisation.

The 'colour RAM' can be accessed at any time for reading or writing. These accesses were also synchronised by the internal clock. The processor 'colour RAM' access had a higher priority than the output generator, therefore, the palette output was delayed for CPU accesses. This could have caused visible effects on the monitor screen if care had not been taken to control the time of updating the colour palette RAM. The standard method of overcoming this difficulty is to only update the colour palette RAM during frame flyback periods. The VSC provided information (VSC Control and Status register) on when the screen was active and when it was in a flyback period so that such synchronisation could occur.

7.5 Summary

The design of one part of the graphics board has been described. The graphics board used the VSC as its graphics controller while the T800 was working as the graphics engine. The T800 was designed as an independent processing node with 1.0 Mbyte of RAM. It was normally booted with the 'Helios' operating system via a link and therefore required no ROM.

The T800 was provided with a coprocessor interface to the VSC which enabled it to read and write picture information into the display memory. This also provided an indirect connection between the SCC68070 and the T800 through the VSC controlled shared memory.

A colour palette BT471 was fitted to provide a good choice of colours for realistic images and to provide output flexibility. The BT471 'colour palette' had 256 x 18 bits of programmable colour palette memory. This gave the programmer a

simple but powerful method of selecting a wide range of colours for particular applications. The BT471 appeared in the T800s memory map to make it accessible directly from 'Helios' rather than using it from the relatively slow SCC68070.

Chapter 8

The parallel graphics

8.1 Introduction

As described in the previous chapter, the T800 was connected to the VSC to act as a high speed coprocessor. This T800 along with the VSC forms a high speed graphics processing node. This was designed keeping in mind that several graphics boards should be able to work together on a picture frame to produce real-time images.

The T800 graphics processing node has its own one Mbytes of memory and was loaded with the 'Helios' operating system. The T800 has access to the graphics memory through the VSC's coprocessor interface. The picture information can be pipelined to the T800 through one or more of its four available links (fig 8.1). This information is processed by the T800 into the required picture file and then put into the VSC's graphics memory. The structure of the picture data file to obtain the screen level parallelism (SLPG) is discussed in a later section.

After the picture data is written into the VSC memory (i.e display memory), the T800 initialises the VSC and the VSC begins producing the pixel and the

synchronisation information for the colour palette and the video monitor.

If more than one graphics node is used, the picture is distributed geometricaly between the available processing nodes. One node is designated as a master node which generates all the synchronisation signals used by the slave boards to generate the video output for the parts of the pictures assigned to them. The VSCs can be assigned the status of a master or a slave by connecting their 'Master/Slave' (M/SN) input pin to VCC or to Ground respectively (fig 8.2). The VSC outputs were combined through a flat ribbon cables connector (mini back plan) on the front of the boards. The VSC master/slave operation is described in the following section.

8.2 VSC master/slave operation

The VSC can be programmed to produced pictures in two colour modes, 4 bit per pixel mode (double mode) or 8-bit per pixel mode (normal mode). In normal mode the picture resolution is halved. A graphics system was designed so that an 8 bit per pixel mode can be obtained from two VSCs in parallel without halving the resolution (chapter 7). The two VSCs working in parallel in this way can produce pictures with 8 bits per pixel in double mode and 16 bits per pixel in normal mode. This increases the range of available colours (from 16 to 256) within double mode. The internal design allows the two VSCs to be connected with their Chip-Select (CSN) inputs shorted together. In this type of arrangement the register of the two VSCs do not overlap. Also different memory banks can be used to avoid display memory overlapping (Appendix A). The master VSC produces the synchronisation signals and both the master and slave contribute to the pixel outputs (fig 8.3). The result is two overlaid picture planes from the

two separate memory banks aligned together to produce a single picture.

In the work undertaken the VSCs were synchronised by two different methods, a serial mode and the parallel mode mentioned earlier. The 8 bit per pixel mode was maintained by the parallel VSCs (fig 8.4) while different parts of the pictures were being generated by different serialised VSCs. The master and slave VSCs in serial mode also work in the same way. That is when one VSC is putting the pixel data out for a given area of the picture, the other VSCs tristate their outputs. The output is still synchronised with the master which generates the required synchronising signals. In this way the picture data is distributed among the VSCs with different VSCs working on different parts of the picture. Each VSC points to the same pixel in the display at the same time, with all VSC outputs tristated except the one which has to contribute that part of the picture. At the output side the two VSCs can be thought of as working in series. One VSC contributing, for instance, the odd lines and the other displaying the even lines. Figure 8.9 showes one scheme to join 'n' number of VSCs together and to distribute the picture among them.

One possibility for exploiting the VSCs in the above mentioned parallel mode was to use a number of VSC chips on one board with a single SCC68070 controlling all of them. Each of the VSC chips could have its own graphics memory and a T800 dedicated as a coprocessor, to process the picture data. A single SCC68070 can control up to eight VSC chips. This is possible by decoding the three unused lines of the SCC68070 address bus (A21- A23), to generate the chip selects for different VSC chips. The SCC68070 is used for initial setup of the VSC and would not be involved in production of graphics data. The picture data flow to the video memory is handled by the transputer, therefore, there is no chance of a 'bottle neck' due to the use of a single relatively slow SCC68070. However, this

approach is not very flexible, as the option of the number of VSCs to be used would be at the printed circuit board level rather than at the application level.

The second approach was to build a fairly general purpose board which can work as a graphics processing node. It would include one VSC, an SCC68070, a T800 and an option for the colour palette. This provided the options for the board to be used as a stand alone graphics node or as part of a multiple node graphics processing system using several boards in parallel. Each board could be connected to the (transputer based) main computer via transputer links. This approach was quite flexible and was adopted. A single board can be used for low speed, high resolution applications as well as being expandable into a parallel graphics processing system depending on the demands of applications.

8.3 VSC synchronisation

VSC chip is designed to operate in three different modes. These are master, slave TV and slave dual modes [17] In the master mode, the VSC generates horizontal, vertical and composite (HSYNCN, VSYNCN and CSYNCN) synchronisation signals. These signals are outputs and can be used to control the video monitor. In this mode the VSC generates the pixel output for the video monitor and controls the synchronisation of these data bytes to the video monitor.

The slave TV mode is used to grab the image information from a slave TV and load it into the memory. The VSC generates the HSYNCN signal. The VSYNCN signal is used as an input from the external source. The VSC expects the digitized pixel information for the 'frame grabbing' on the memory data bus. The picture is loaded into the memory synchronised with the VSYNCN and the

VSC displays the picture along with the frame grabbing action. During frame grabbing the FG bit in the CSR register is set (Appendix C) and no access to the CPU or coprocessor is granted. One or two frames are grabbed depending upon whether interlace or non-interlace mode has been selected. At the end of the image grabbing, the FG bit in the CSR register is reset automatically. The CSYNCN signal is generated for the use of the external 'Phase Lock Loop' (PLL). In this case the CSYNCN is a 50% duty cycle HSYNCN signal with high during the first half of the line. This option was not provided on the graphics processing board.

The slave dual mode is used for the slave VSC chips for the master/slave operation. In this mode the HSYNCN and the VSYNCN pins become the timing inputs. When the display is disabled (i.e. DE = 0 in the CSR register), the CSYNCN is an input to initialise the synchronisation mode. During this mode, if this is pulled low (fig 8.2), the slave dual mode is selected [17]. When the display is enabled (i.e. DE = 1), this CSYNCN signals becomes an output and generates the phase error between the external HSYNCN (i.e. from the master) and the internal HSYNCN.

In order for the VSCs to work together in the above mentioned master/slave mode, the clock must be common and connected to all 'XTAL1' pins. The common clock is necessary to synchronise the internal functions of VSCs. The reset is also synchronised to avoid any phase errors between the internal clocks. A suggested circuit [17] was used. The XTAL2 outputs of all the VSCs are interconnected to synchronised the latched reset. This unusual method of commoning the CMOS outputs (fig 8.5) works because the rising edge of the first clock triggers a D-type flip-flop and the reset is latched. This latched reset, connected to all RSTINN inputs, forces the clock output to synchronise. The VSCs, after this,

remain synchronized thus effectively producing no loading effects on the shorted outputs.

The display files can be organized to tristate the pixel outputs (P0-P7) and to enable them later whenever required. The possible display file organisation is discussed in section 8.4. If care is taken ensure that no two outputs are enabled at the same time, then the pixel outputs can remain commoned on the ribbon-cable back plane. A total of thirteen signals (excluding ground) are available on this ribbon cable (fig 8.6).

A universal board was designed. The sockets for colour palette, clock module and the reset circuit including D-type latch were provided on each board. The function of either master or slave is selected through a set of jumpers. These jumpers and their interconnections for master and slave boards are given in fig 8.6.

8.4 Display files and picture distribution

The VSC can handle 3 types of display files. These are called normal (or bitmap), MOSAIC and run-length files. In a bitmap file each pixel has its own address and an 8 bit value in the display memory. The memory contents are output as pixel data without any further processing. The MOSAIC and run-length file formats have compressed information. They can be used to save space and can achieve faster speed for picture loading and animation.

In a MOSAIC file the information is compressed by a factor of 'n x m' where n is the horizontal MOSAIC factor and m is the vertical MOSAIC factor [17]

The horizontal MOSAIC factors of 2, 4, 8, and 16 are possible (fig 8.7). Depending upon the horizontal MOSAIC factor the VSC repeats the same pixel for the required number of times on the line. When the line is finished, the display continues on the next line. The vertical MOSAIC factor is independent of the horizontal MOSAIC factor. This vertical MOSAIC factor can be achieved through the DCA by duplicating the line. The Video Start Register (VSR) can be loaded with the memory address of the previous line and the VSC will repeat the line. The 'Reload' instruction can be repeated to obtain the required vertical MOSAIC factor.

In the run-length format, the file is organised as 7 bits of pixel colour where as the Most Significant Bit (MSB) of the pixel location is used as a flag to define the type of the stored information as either A or B (fig 8.7). The type A defines the colour of a single pixel. This pixel is output once and is not repeated. In type B, consecutive pixels with the same colour can be grouped together in a 16 bit word. The first byte defines the type and the pixel colour while the second byte gives the number of pixel repetition. The VSC sees the 'type flag' and outputs the same pixel for the required number of times. The pixel repetition field can be defined between 2 and 255. If this field is zero, the VSC ends the line with the same colour, and starts the next line with the next 'information word'. In the run-length file format, the colour is defined by 7 bits in the normal display mode and by 3 bits in the double mode. This limits the range of available colours from 256 to 128 in the normal mode. The MSB in this case, is always suppressed for the pixel output.

The display files can be further manipulated by the Image Control Area (ICA) and the Dynamic Control Area (DCA) (fig 8.8). These two areas can be selectively enabled or disabled through the DCR register. The ICA always starts at address

'400H' and is of variable length. This is accessed during the fly back period of the picture and is terminated by a 'stop' or a 'Reload VSR and stop' (RVSR) instruction. The DCA, on the other hand is organised in rows that is at the end of each display line. It can either be placed consecutively after each display line in memory or can be independent, arranged somewhere away from the picture buffer. The DCP register is used to point to the DCA. The length of the DCA is fixed and can be selected as either 16 bytes or 64 bytes. It is long word aligned and the 'DCA instructions' are fetched during the horizontal retrace period. The instruction set for the ICA and DCA are given in Appendix B.

Because the ICA and DCA offer the facility of fetching and using control information, a display file can be organised which can control precisely the pixel output of VSCs in the previously suggested parallel mode by using the reload parameter commands in the DCA (Appendix B). The DCR register can be reprogrammed to enable or disable the pixel output from the VSC at the end of the display line (fig 8.9). The 'file mode' can also be changed, that is bit map, MOSAIC or run-length file modes can be selected.

In the multi-VSC system the VSCs can be arranged to display the lines on rotation basis. For example, if VSC1 is displaying the first line and if 'n' VSCs are used, the next active line for VSC1 will be the (n+1)th line. The output of VSC1 can be disabled at the end of the first line and can remain disabled for n-1 lines (fig 8.9). At the end of the nth line the output can be enabled again to display the (n+1)th line. The 2nd, 3rd.....nth display lines will be the inactive lines for VSC1 therefore the output must remain disabled. These inactive lines though not being displayed still reside in the memory.

One way of using memory engaged in the inactive lines is to hold information for

the next frame in the picture. The DCA information can be modified to enable the 2nd line for the 2nd frame while the first line becomes inactive. In this way 'n' frames can be built in parallel for an 'n VSC system' in the same memory area. The disadvantage of this would be that the VSC will be using the display window for pixel output thus unnecessarily occupying the memory bandwidth. This is because the inactive lines are fetched even when they are not being displayed. This reduces the time available for the coprocessor to update the display memory for the next frame.

An alternative approach is to store the inactive lines in the run-length file type B form. In this type if the second byte in the run-length information is zero then the line is completed with the same colour. If the inactive lines are stored in run-length file format, a single word could represent the whole line (fig 8.10). At the end of the active line, the DCRs can be reloaded to read the next line in run-length mode. The VSC would access the next word and finish off the line without requiring any further access to the picture buffer hence saving the time for coprocessor accesses. The graphics processing system can work faster.

In the run-length file arrangement, however, the video memory becomes irregular. Therefore only an independent DCA can be used. The active lines must not be stored in run-length format because this will reduce the colour range (128 instead of 256 colours which are available in bitmap or MOSAIC file modes). Therefore the active lines are better if stored in the bitmap file format.

8.5 Summary

The parallel graphics system to be used with a parallel processing system has been described . The SCN66470 (VSC) was used as a video controller. The VSCs were made to work in parallel thus sharing the picture output workload. They were made to work as master and slave(s) producing synchronised video output.

A universal board was designed to be used either as a master or slave board in a parallel graphics system. It is shown that the display files can be organised to use the minimum memory and also reduce the number of RAM accesses. This will save a few extra micro seconds thus giving the T800 coprocessor a better opportunity for picture updating.

Chapter 9

Conclusions

A general purpose Input/Output system and a versatile graphics system have been implemented. The I/O board was built to include all the standard facilities, including both the hard and floppy disks, a printer interface and the serial port for the console interface. A clock/calendar chip was also added on the Inter-Integrated Circuit (I2C) bus to facilitate the time/date backup.

The I/O board was connected with one hard disk unit. The designed boards, however, can be used in parallel with more than one disk to provide faster more efficient auxiliary storage service. A method for providing parallel storage facilities has been suggested. If used, this can provide much faster data movement when compared with the situation if a single disk is used.

The graphics board was implemented to work as either a stand-alone graphics processing system or to be used as a part of the parallel graphics system. The Screen Level Parallel Graphics (SLPG) technique was used. This works faster compared to a graphics system using a single device outputting the video signals. It is shown that with the designed board, the graphics system can be expanded to achieve the required speed by distributing the picture on the screen level.

A display file organisation to achieve screen level parallelism has been described. It is shown that with the suggested file the picture memory accesses from the VSC can be reduced to 'one access' per non-active display lines of the picture. The lines can be activated or disabled by either enabling or disabling respectively the VSC pixel output at the end of the previous line. Moreover, the multiple boards can be put together to work on a single frame and the parallelism can be achieved up to the level of the individual lines within the picture.

The T800 transputer, used as the graphics processor, was interfaced to the VSC to transfer the picture information to the display memory using the VSC coprocessor handshake signals. It uses the 'CPU/coprocessor' window of the VSC memory cycle. When the VSC is operating at 30 MHz, the coprocessor window exists every (16 cycles / 30 MHz) 532.8 nS. Assuming that the SCC68070 does not access the memory, that is every window is granted to the T800, and also that the T800 is making a 16 bit word transfer in every display memory cycle, a full frame must be updated every (768 pixels x 560 lines x 532.8 nS / 2 bytes) 114.57 mS. For the real-time animation, the screen must be updated about 30 times a second, that is one frame to be updated every 33.3 mS. This speed can be achieved by adding extra boards to the system. In this case, the graphics system will require (114.57 mS / 33.3 mS = 3.44, that is) four boards.

The above calculations were made on the assumption that the SCC68070 is not accessing the memory and also that the T800 is always ready to make data transfers. In actual case, the situation will be different and the system may require more than four boards.

Chapter 10

Further work

The I/O system was implemented for a single hard disk unit which was also serving as the system central storage device (i.e. the main hard disk in the system). The related software, i.e. device drivers, were also developed accordingly. This I/O system can be expanded and the device drivers can be modified to accommodate the idea of the Combined Disk Arrangement (CDA). If implemented, this arrangement must be able to offer a speed compatable to the multiprocessor system requirements. Also, the provision for a faster virtual memory can be made by providing a number of hard disk units to the clusters of processing nodes.

The parallel graphics system allowed the use of VSC in the double resolution mode (i.e. 4 bits per pixel mode). This implied that two VSCs were to be used in parallel to provide the required 8 bits per pixel output for a wider colour range. The two VSCs, at present, reside on two separate boards which unnecessarily occupies one extra slot in the backplane of the host computer. The hardware design of the graphics board can be modified and two VSCs can be put on a single board thus freeing the backplane slot. Moreover, one processor (i.e. SCC68070) can be used for initialising both the VSCs.

The SCC68070s on the graphics boards are presently involved in initialising of VSCs only. After the initial setup, they remain inactive throughout the picture generating and displaying process. If the workload of these processors is increased, then, due to the repetitive accesses to the 'video and system' memory, the graphics memory bandwidth available to the T800 is reduced considerably. The processor can, however, be given some slower jobs, for example data handling on the serial ports (i.e. RS-232 and I2C buses) for another console, a mouse or the local area network. This type of applications, if the RAM accesses can be kept to a minimum by running the port handling routines off the ROM, will not seriously effect the graphics memory bandwidth for the T800. The related port handling routines can be developed and the serial ports on the SCC68070 can be utilised.

The graphics board includes the T800 transputer as a graphics engine. Because of the higher T800 throughput, the board can be used for more than just transferring picture data to the display memory. Low level graphics routines, such as line drawing area filling etc, can be developed and placed on the graphics board. The host processor can derive the initial picture coordinates, place them into a packet and then the packet can be passed to all the slave (i.e. graphics) processors in parallel. Each T800 can derive information from the packet related to the part of the picture allocated to it. This will eliminate the need to provide each of the T800 with the specific information related to its part of the picture.

References

1. 'i860 Data Sheet', Intel corporation, 1988.
2. '16/32-bit Highly-integrated microprocessor SCC68070',
   User manual, Part 1-Hardware,
   Philips Components Ltd., 1988.
3. GIBSON, D.H. and W.L. SHEVEL.:
   'Cache turn up a treasure.'
   Electronics, Oct 13, 1969, pp.105-107.
4. MONROE, R.N.:
   'Add-in cache memory doubles minicomputer processing speed',
   Computer Design, Oct. 1979, pp.115-120.
5. EDEN, R.C., B.M. WELCH, R. ZUCCA and S.I. LONG.:
   'The prospects for ultra high-speed VLSI GaAs digital logic',
   IEEE J. of S.S. Circuits., Vol SC-14, No. 2, Apr. 1979, pp.221-239.
6. LINEBACK, J.R.:
   'Feature: GaAs RISC processor is in the work',
   Electronics, June 9, 1986, pp.21-22.
7. GOUTZOULIS, A.P. and W.T. RHODES.:
   'Digital electronics meets its match',
   IEEE Spectrum, Vol. 25, No. 8, 1988, pp.21-25.
8. MIRSALEHI, M.M. and M.A.G. ABUSHAGUR.:
   'Optical and optoelectronic computing' in
   'Advances in computer',
   Edited by M.C. YOVITS.,
   Academic Press Inc., 1989.
9. MILUTINOVIC, V., N.L. BENITZ and K. HWANG.:
   'A GaAs-based microprocessor architecture for real-time applications',
   IEEE Trans. on Computers., Vol, C-36, No. 6, June 1987, pp.714-727.
10. NGOH, L.H. and T.P. HOPKINS.:
    'Transport protocol requirements for distributed multimedia information system',
    The Computer J., Vol. 32 No. 3, 1989.

11. BHANDARKAR, D.P.:
'Analysis of memory interface in multiprocessors',
IEEE Trans. on Computers., Vol. C-24,No. 9,Sep. 1975, pp.897-908.

12. FLYNN, M.J.:
'Very high-speed computing systems',
Proc.IEEE, Vol. 54, Dec. 1966, pp.1901-1909.

13. CHU, W.W.:
'Optimal file allocation in a multiple computer system',
IEEE Trans. on computers, Vol. C-18, No. 10, Oct. 1969, pp.885-889.

14. WILKES, M.V.:
'Slave memories and dynamic storage allocation',
IEEE Trans. on Electronic Computers, Apr. 1965, pp.270-271.

15. BELADY, L.A. and C.J. KUEHNER.:
'Dynamic space sharing in computer systems',
ACM Comm., Vol. 12, 1968, pp.282-288.

16. KUROSE, J.F. and R. SIMHA:
'A microeconomic approach to optimal resource
allocation in distributed computer systems',
IEEE Trans. on Reliability, Vol. 38, No. 5, May 1989, pp.705-717.

17. 'SCN66470-Video and System Controller (VSC)',
User reference manual,
Philips components Ltd, 1988.

18. 'IMS T800 Data Sheet', INMOS 1987.

19. '8096KB, Embedded 32-bit microprocessor
with integrated floating-point unit',
Preliminary information,
Intel Corporation, 1988.

20. BOCHMANN, G.V. and C.A. SUNSHINE.:
'Formal methods in communication protocol design',
IEEE Trans. on Comm., Vol. COM-28, Apr. 1980, pp.624-631.

21. WITTIE, L.D.:
'Communication structures for large networks of microcomputers',
IEEE Trans. on Computers, Vol. C-30, Apr. 1981, pp.264-273.

22. YEN, D.W.L., J.H. PATEL, and E.S. DAVIDSON.:
'Memory interference in synchronous multiprocessor systems',
IEEE Trans. on Computers, Vol. C-31, No. 11, Nov. 1982, pp.1116-1121.

23. DAS, C.R. and L.N. BHUYAN.:
'Bandwidth availability of multiple-bus multiprocessors',
IEEE Trans. on Computers, Vol. C-34, No. 10, Oct. 1985, pp.918-926.

24. TOWSLEY, D.:
'Approximate models of multiple-bus multiprocessor systems',
IEEE Trans. on Computers, Vol. C-35, No. 3, Mar. 1986, pp.220-228.

25. MARSDEN, B.W.:
'Communication network protocols',
Chartwell-Bratt (publishing and training) Ltd., England 1986.

26. HOROWITZ, E. and A. ZORAT.:
'The binary tree as an interconnection network: Applications
to multiprocessor systems and VLSI',
IEEE Trans. on Computers, Vol. C-30, No. 4, Apr. 1981, pp.247-253.

27. PEASE, M.C.:
'The indirect binary n-cube microprocessor arrays',
IEEE Trans. on Computers, Vol. C-26, No. 5, May 1977, pp.458-471.

28. CLYMER, B.D. and J.W. GOODMAN.:
'Optical clock distribution to silicon chips',
Opt. Eng., Vol. 25, 1986, pp.1103-1108.

29. WHITE, S.W., N.R. STRADER and V.T. RHYNE.:
'A VLSI-based I/O formatting device',
IEEE Trans. on Computers, Vol. C-33, No. 2, Feb. 1984, pp.140-149.

30. BROOKER, R.A.:
'Some techniques for dealing with two level storage',
Computer Journal, Vol. 2, 1960.

31. NASRAOUI, A.:
'Parallel coding of binary images',
Computing, Vol. 41, 1989, pp.1-17.

32. ZAKI, M. and M.M. ELBORAEY.:
'Analysis of reliability models for interconnecting MIMD systems',
The Computer Journal, Vol. C-31, No. 4, 1988, pp.304-311.

33. 'Programmable Logic' Handbook/Data book,
Advanced Micro Devices, 1987.

34. Philips data handbook,
'Semi-custom Programmable Logic Devices (PLD)',
Philips components Ltd., 1987.

35. BACON, J.:
'The MOTOROLA MC68000: An introduction to processor, memory and interfacing',
Prentice/Hall Internaional (UK) Ltd., 1986.

36. 'Tripos Programmer's reference manual',
Metacomco, issue may 1986.

37. 'Microcore: An evaluation board for 68070 and VSC',
Philips components Ltd., April 1987.

38. 'SCN68454: Intelligent Multiple Disk Controller (IMDC)' Data sheet,
Mullard Ltd., july 1985.

39. 'WD279X-02 Floppy disk formatter/controller Family', Data book,
Western digital corporation, 1986.

40. 'DP8474 data sheet',
National Semiconductors Ltd., 1986.

41. 'SCB68459: Disk Phase-Locked Loop (DPLL)' Data sheet,
Mullard Ltd., july 1986.

42. 'EDWARDS, B.:
'SCSI drives peripherals market to new highs',
Computer design, Dec 1985, pp. 73.

43. American National Standards Institute,:
'Small Computer Systems Interface (SCSI)',
Approved June 23, 1986.

44. Mullard.:
'I2C bus specifications',
Mullard Ltd., July 1985.

45. Mullard:
'I2C bus and compatible components for flexible control systems',
Mullard Ltd., Nov. 1985.

46. 'Introduction to Tripos',
Metacomco, issue may 1986.

47. 'Rodime RO650(C) series disk drives:
Product specifications (Rev 1),
Rodime Europe Ltd, 1987.

48. Philips components technical handbook,
'I2C-bus compatible ICs',
Philips components Ltd., 1988.

49. 'IMS C012 link adaptor',
'Data Sheet', INMOS 1987.

50. 'Tripos Technical reference manual',
Metacomco, issue may 1986.

51. 'AMAZE, Software package for PLD design':
Philips components Ltd., 1987.

52. 'Memory devices data book', Toshiba, 1987.

53. 'Helios operating system', Perihelion software,
Printice Hall, 1989.

54. 'BT471; 256 Color Palette',
Preliminary information,
Brooktree Corporation, San Diego, CA.

Appendix A

VSC controlled memory map for 256K DRAM devices.

Address	Master or Slave TV mode	Slave Dual mode
00 0000 07 FFFF	Bank 1 0.5 Mbytes	Bank 1 0.5 Mbytes
08 0000 0F FFFF	Bank 2 0.5 Mbytes	Bank 2 0.5 Mbytes
10 0000 17 FFFF	Bank 3 0.5 Mbytes	Bank 3 0.5 Mbytes
18 0000 1F FBFF	System ROM 128 Kbytes	Not used
1F FC00 1F FF7F	System I/O	System I/O
1F FF80 1F FFBF	DRAM I/O	Not used
1F FFC0 1F FFDF	Not used	VSC internal registers
1F FFE0 1F FFFF	VSC internal registers	Not used

Appendix B

ICA and DCA instruction set

Instruction (hex)	Acronym	Function
0XXX XXXX	STOP	The instruction fetches from ICD/DCA are stoped.
1XXX XXXX	NOP	No operation
2XXP PPPP	RDCP	Reload DCP with the specified address.
3XXP PPPP	RDCP and STOP	Reload DCP with the specified address and then 'stop'.
4XXP PPPP	RVSR	Reload VSR with the specified address.
5XXP PPPP	RVSR and STOP	Reload VSR with the specified address and then 'stop'.
6XXX XXXX	Interrupt	Generates an interrupt for the CPU.
70PP XXXX	RBCR	Reload border colour register with the specified colour.
78PP XMMM	RBCR and Display parameters	Border colour register is loaded with colour 'PP' and different bits in DCR and DCR2.

		MMM = -a-b -cd- efgh, where a = SS, b = CM, c = OM1, d = OM2 e = MF1, f = MF2, g = FT1 and h = FT2.
V000 0000	BEP control	Control word for Back End processor. All 32 bits are output on the pixel bus (P0-P7). (V = 1XXX binary)

X = don't care.

Appendix C

VSC registers.

Control and Status Register (CSR).

Address = 1FFFE0

Status = Write only

Bit Number	Acronym	Function
B0	BE	Bus Error enable (for watch dog timer).
B1	ST	Standard (0 = 50 Hz, 1 = 60 Hzs)
B2	ED	Early DTACKN (two clock periods)
B3	DD	Enable DTACKN Delay (delay is controlled by DD1 and DD2).
B4	CG	Character Generation. Activates DTACKN timings for CGROM.
B5	TD	Type of DRAM devices (0 = 64K and 1 = 256K DRAM devices)
B6	DM2	DRAM access Mode. The access can be
B7	DM1	configured between DM1 and DM2 as

		0 = Normal, 1 = Page, 2 = Nibble and 3 = Dual Port mode.
B8	DD2	DTACKN Delay. DD2 and DD1 can be
B9	DD1	programmed to delay DTACKN for ROM as 0 = 8 to 10, 1 = 4 to 6, 2 = 6 to 8 and 3 = 12 to 14 VSC clock periods.
B10	EW	Early Write (for early DTACKN generation)
B11	N/A	Not Used.
B12	N/A	Not Used.
B13	N/A	Not Used.
B14	DI2	Disable Interrupt (from pixel accelerator).
B15	DI1	Disable Interrupt (from video generator).

Status Register (SR).

Address = 1FFFE1

Status = Read only

Bit Number	Acronym	Function
B0	BE	Bus Error.
B1	IT1	Interrupt from display generator.
B2	IT2	Interrupt from Pixel Accelerator (PIXAC).
B3	N/A	Not Used.
B4	N/A	Not Used.
B5	PA	Frame Parity (for interlace pictures only).
B6	FG	Frame Grab in progress.
B7	DA	Display Active.

Display Command Register (DCR).

Address = 1FFFE2

Status = Read/Write

Bit Number	Acronym	Function
B0	A16	Address For Video Start Register.
B1	A17	Address For Video Start Register.
B2	A18	Address For Video Start Register.
B3	A19	Address For Video Start Register.
B4	DC	Selectively enable the ICA and
B5	IC	DCA as 0 = Disable ICA and DCA 1 = Enable ICA and Reduced DCA 2 = Enable ICA but disable DCA 3 = Enable ICA and full DCA
B6	DF	Double Frequency (used with B11)
B7	FG	Frame grabbing in progress.
B8	CM	Colour Mode 0 = 8 bits per pixel 1 = 4 bits per pixel
B9	LS	Enable logical screen (512 bytes per line)
B10	SS	Enable Full screen (with no boarders)
B11	SM	Scan Mode selection (used with B6 as follows) 0 = non interlace mode 1 = Double frequency mode 2 = Interlace mode 3 = Interlace field repeat mode (two interlaced frames

showing same memory)

B12	FD	Freame Duration (0 = 50 Hz, 1 = 60 Hz)
B13	CF2	Crystal frequency reference.
B14	CF1	It must be programmed as 0 = 20 Mhz, 1 = 24 Mhz, 2 = 28 Mhz and 3 = 30 Mhz crystal frequency.
B15	DE	Display Enable.

Display Command Register number 2 (DCR2).

Address = 1FFFE8

Status = Read/Write

Bit Number	Acronym	Function
B0	A16	Address For DCA pointer Register.
B1	A17	Address For DCA pointer Register.
B2	A18	Address For DCA pointer Register.
B3	A19	Address For DCA pointer Register.
B4	N/A	Not used
B5	N/A	Not used
B6	N/A	Not used
B7	N/A	Not used
B8	FT2	File Type. Selected as
B9	FT1	0 = Bitmap file format 1 = Bitmap file format 2 = Run-length file format 3 = MOSAIC file format
B10	MF2	Horisontal MOSAIC Factor.
B11	MF1	(can be programmed for 0 = 2, 1 = 4, 2 = 8 and 3 = 16 MOSAIC factor)
B12	ID	Enable Independent DCA.
B13	OM2	Output Mode (for selecting the
B14	OM1	appropriate nibbles in double mode)
B15	N/A	Not used

Video Start Register (VSR).

Address = 1FFFE4

Status = Read/Write

Bit Number	Acronym	
B0-B15	A0-A14	Video start address. Along with the lower nibble of the DCR, this register provides 20 bit current display address.

DCA Pointer register (DCP).

Address = 1FFFEA

Status = Read/Write

Bit Number	Acronym	Function
B0-B1	N/A	Not used
B2-B15	A2-A15	DCA address. Along with the lower nibble of the DCR2, this register provides the address of the DCA.

Border Colour Register (BCR).

Address = 1FFFE7

Status = Read/Write

Bit Number	Acronym	Function
B0-B7	BC0-BC7	Border colour. The selected boreder colour can be displayed for normal mode (B0-B7) or double mode (B4-B7).

Appendix D

PAL/PLD coding

Symbols

*	AND function
+	OR function
/	Prefix to show negation
;	Start of comment
GND	Ground
VCC	Supply voltage
NCx	Pin not connected

Inputs and outputs are defined active high or low by name.

PAL for WD279X FDC

```
PAL16L8                                          PAL DESIGN SPECIFICATION

BUFCON                                                              MH

A3 A4 A5 A6 A7 A8 A9 LDSN CSION GND

RWN CSN DEL2 DEL4 DWN DRN WEN REN DTACKN VCC



/REN = /A3 * /A4 * /A5 * /A6 * /A7 * /A8 * /A9 * RWN * DLY2
                                          ;enable read

/WEN = /A3 * /A4 * /A5 * /A6 * /A7 * /A8 * /A9 * /RWN * DLY2
                                          ;enable write

/DWN = A3 * /A4 * /A5 * /A6 * /A7 * /A8 * /A9 * /RWN
                                          ;enable mask register write

/DRN = A3 * /A4 * /A5 * /A6 * /A7 * /A8 * /A9 * RWN
                                          ;enable mask register read

DTACKN = REN * WEN * DWN * DRN            ;acknowlege for read/write

         + /DLY4                          ;end DTACKN


This pal decodes the signal for WD279X floppy disk controller.
```

PLD for DP8474 FDC

```
PLS153                                          PLD DESIGN SPECIFICATION

BUFCON                                                          MH

CSION DONEN RESETN A9 RWN DRQ INT NCO CSN GND

TC RESET RDN IOSEL WRN A8 INT1N REQ2N DTACKN VCC


DTACKN = /IOSEL                 ;tristated output
                                ;enabled with address decode

REQ2N  = /DRQ                   ;invert DRQ

TC     = /DONEN                 ;invert DONEN

INT1N  = /INT                   ;tristated output
                                ;enabled with /INT

RESET  = /RESETN                ;invert RESETN

/RD    = /(/CSION * RWN)        ;read enable

/WR    = /(/CSION * /RWN)       ;write enable

IOSEL  = A8 * A9 * CSION        ;address decode

/CS    = /IOSEL


This PLD provides address decode and other conterol signals

for DP8474 floppy disk controller.
```

PLD for SASI interface controller

```
PLHS501                                              PLD DESIGN SPECIFICATION

BUFCON                                                                    MH

;External pin names are same as in the circuit diagram

Internal Nodes:

MSK, REN, WEN, WEN1, DMAREQ, LUDS, DMACLK,

DMARD, RSET, INT, Q0, Q1, Q2, Q3, Q4, Q5, Q6


I/O direction:

DB5 = (/LDSN * /CSION * /A7 * A8 * /A9)                ;dtack enable

DB6 = /INT                                             ;interrupt enable


OE0 = 1                                                ;output enable

OE1 = 1                                                ;output enable

OE2 = /RSET                                            ;SASI reset enable

OE3 = Q0                                               ;SASI select signal


XE0 = (/A1 * /A7 * A8 * /A9 * /CSION * RWN * /LDSN)    ;status register

XE1 = (/A1 * /A7 * A8 * /A9 * /CSION * RWN * /LDSN)    ;read enable

XE2 = (/A1 * /A7 * A8 * /A9 * /CSION * RWN * /LDSN)    ;signals


I/O steering:

S0, S1, S2, S3 = 0                                     ;all are outputs


Logic equations:

MSK = (/A1 * /A7 * A8 * /A9 * /CSION * /RWN * /LDSN)   ;address decode
                                                       ;for mask register

WEN = /(A1 * /A7 * A8 * /A9 * /CSION * /RWN * /LDSN)   ;write enable

REN = /(A1 * /A7 * A8 * /A9 * /CSION * RWN * /LDSN)    ;read enable
```

```
;mask register (7 bit latch)

Q0 = (D0 * MSK * RESETN) + (/(MSK * /D0) * RESETN * Q0)

Q1 = (D1 * MSK * RESETN) + (/(MSK * /D1) * RESETN * Q1)

Q2 = (D2 * MSK * RESETN) + (/(MSK * /D2) * RESETN * Q2)

Q3 = (D3 * MSK * RESETN) + (/(MSK * /D3) * RESETN * Q3)

Q4 = (D4 * MSK * RESETN) + (/(MSK * /D4) * RESETN * Q4)

Q5 = (D5 * MSK * RESETN) + (/(MSK * /D5) * RESETN * Q5)

Q6 = (D6 * MSK * RESETN) + (/(MSK * /D6) * RESETN * Q6)


INT = /(Q6 * (MSG + Q5) * (C/D + Q4) * (/C/D + Q3) * /REQ)
                                                    ;interrupt enable

INT2N = 0                                           ;interruput output

DTACKN = 0                                          ;DTACKN output

RSET = (/Q1 * RESETN)                               ;external reset
                                                    ;and reset from mask
                                                    ;register

RST = /(1)                                          ;SASI reset

CLK7 = /( /(/(A1 */A7 * A8 * /A9 * /CSION * /LDSN) * ACK1N))
                                                    ;SASI acknowledge

WEN1 = /(I/O * /REQ)

DMARD = /(/RWN * /ACK1N)

HENN = REN * (WEN1 + LUDS) * DMARD                  ;high byte enable

LENN = REN * (WEN1 + /LUDS) * /Q0 * DMARD           ;low byte enable

WDATA = /(WEN * /(RWN * /DTACKN * ACK1N))           ;write buffer enable

RDATA = /REQ * /I/O                                 ;read buffer enable

IOOUT  : XR1 = I/O
                                                    ;status register D0
         XR2 = 0

CDOUT  : XR1 = C/D
                                                    ;status register D1
         XR2 = 0

MSGOUT : XR1 = MSG                                  ;status register D2
```

```
         XR2 = 0

BSYOUT : XR1 = BSY                                    ;status register D3

         XR2 = 0

REQOUT : XR1 = REQ                                    ;status register D4

         XR2 = 0


LUDS = (DMACLK * UDSN)

       + (/(DMACLK * /UDSN) * LUDS) + /WEN            ;latch for USDN

DMACLK = RWN * DTACKN * /ACK1N * /(LDSN * UDSN)       ;clock for UDSN latch


CLK8 = /(/(Q2 * C/D * /REQ))                          ;clock for DMA request

REQACK = ACK1N * RESETN                               ;clear DMA request

DESCRIPTION:

This PLD provides the mask register, the status register and other control

signals for SASI interface.
```

PLDs for the graphics board

PLD for the address decode

```
PLHS153                                                PLD DESIGN SPECIFICATION

BUFCON                                                                  MH/VSG

A24 A25 A26 A27 A28 A29 A30 A31 /RF GND

/S3 /S1 NC0 /WAIT /PLTEN /VSCMEM NC1 /LCAS /LRAS VCC


/LRAS = /(S1 * A31 * /A30 * /A29 * /A28 * /A27 * /A26 */A25 * /A24

        + S1 * RF)                      ;RASN decode for T800 local DRAM

/LCAS = /(S3 * /RF * A31 * /A30 * /A29 * /A28 * /A27 * /A26 */A25 * /A24)
                                        ;CASN decode for T800 local DRAM

/VSCMEM = /(/RF * A31 * /A30 * /A29 * /A28 * /A27 * /A26 */A25 * /A24)
                                        ;display memory access decode

/PLIEN = /(/RF * A31 * /A30 * /A29 * /A28 * /A27 * /A26 */A25 * A24)
                                        ;colour palette access decode

/WAIT = /(/RF * A31 * /A30 * /A29 * /A28 * /A27 * /A26 */A25)
                                        ;wait decode


DESCRIPTION:

Listing of U13 (PLHS153) providing the

memory address decode on the graphics board.
```

PLD for the next state decode

```
PLS153                                          PLD DESIGN SPECIFICATION

BUFCON                                                          MH/VSG

/RAS /CYACK /WL /WH MEMWAIT S0 S1 S2 S3 GND

S4 NS4 NS3 NS2 NS1 NS0 NC0 NC1 /RESET VCC



NS0 = /S0 * /S1 * /S2 * /S3 * MEMWAIT * /RESET

     + S0 * /S1 * /S2 * /S3 * /RESET

     + /S0 * /S1 * /S2 * S3 * /RESET;

NS1 = S0 * /S1 * /S2 * /S3 * CYACK * /RESET

     + S0 * S1 * /S2 * /S3 * /RESET

     + /S0 * S1 * /S2 * /S3 * /RESET

NS2 = /S0 * S1 * /S2 * /S3 * RAS * /WL * /WH * /RESET

     + /S0 * S1 * S2 * /S3 * /RESET;

NS3 = /S0 * S1 * /S2 * /S3 * RAS * WL * WH * /RESET

     + /S0 * S1 * /S2 * /S3 * RAS * WL * /WH * /RESET

     + /S0 * S1 * /S2 * S3 * /RESET

     + /S0 * /S1 * S2 * /S3 * /RESET

     + /S0 * /S1 * /S2 * S3 * /RESET;


DESCRIPTION:

Listing of U27 (PLS153) providing the next state decode.
```

PLD for coprocessor access control

```
PLHS153                                          PLD DESIGN SPECIFICATION

BUFCON                                                          MH/VSG

/RAS /CYACK /WL /WH MEMWAIT S0 S1 S2 S3 GND

S4 /COPRD /GBA /INFOEN /COPWR /WAITOFF /CYREQ NC0 NC1 VCC


/CYREQ = /( /S0 * /S1 * /S2 * /S3 * MEMWAIT

         + S0 * /S1 * /S2 * /S3

         + S0 * S1 * /S2 * /S3

         + /S0 * S1 * /S2 * /S3 * /RAS )

WAITOFF =  /S0 * S1 * /S2 * /S3 * RAS * WL * WH

         + /S0 * S1 * /S2 * /S3 * RAS * WL * /WH

         + /S0 * S1 * /S2 * /S3 * RAS * /WL * WH

         + /S0 * S1 * S2 * /S3

         + /S0 * /S1 * S2 * /S3

         + /S0 * S1 * /S2 * S3

         + /S0 * /S1 * /S2 * S3

/INFOEN = /( S0 * S1 * /S2 * /S3

         + /S0 * S1 * /S2 * /S3 * /RAS )

/GBA = /( /S0 * S1 * /S2 * /S3 * RAS * WL * WH

         + /S0 * S1 * /S2 * /S3 * RAS * /WL * WH

         + /S0 * S1 * /S2 * /S3 * RAS * WL * /WH )

/COPWR = /( /S0 * S1 * /S2 * /S3 * RAS * WL * WH

         + /S0 * S1 * /S2 * /S3 * RAS * /WL * WH

         + /S0 * S1 * /S2 * /S3 * RAS * WL * /WH

         + /S0 * S1 * /S2 * S3 )
```

```
/COPWR = /( /S0 * S1 * /S2 * /S3 * RAS * /WL * /WH
         + /S0 * S1 * S2 * /S3 )

DESCRIPTION:
Listing of U28 (PLHS153) providing the
display memory access control for coprocessor.
```

PAL for colour palette access control

```
PAL16R4                                                PLD DESIGN SPECIFICATION

BUFCON                                                                  MH/VSG

PCLK NC0 NC1 /MS0 /RD /WB0 /PLTEN NC2 NC3 GND

OE /PRD /PWR /EN /S2 /S1 /S0 RS1 /WAITOFF VCC

ENABLE (WAITOFF)                          ;enable output if WAITOFF

ENABLE (PWR)                              ;enable output if PWR

ENABLE (PRD)                              ;enable output if PRD

EN = PLTEN * MS0                          ;palette access decode

S0 = /S0 * /S1 * /S2 * EN                 ;latch access
    + S0 * /S1 * /S2

S1 = S0 * /S1 * /S2 * /EN
    + S0 * S1 * /S2
    + /S0 * S1 * /S2

S2 = /S0 * S1 * /S2
    + /S0 * S1 * S2


WAITOFF = S0 * /S1 * /S2 * EN

PWR = /S0 * /S1 * /S2 * WB0 * EN
    + S0 * /S1 * /S2 * WB0 * EN
PRD = /S0 * /S1 * /S2 * RD * EN
    + S0 * /S1 * /S2 * RD * EN

DESCRIPTION:
```

Listing of U33 (PAL16R4) providing access

to colour palette registers on the graphics board.

PAL for nibble access control

```
PAL16R4                                          PLD DESIGN SPECIFICATION

BUFCON                                                          MH/VSG

/CAS0 /CAS1 /WB3 /WB2 /WB1 /WB0 /COPWR /COPRD NC0 GND

NC1 /CAS1D /CAS1C /CAS1B /CAS1A /CAS0D /CAS0C /CAS0B /CAS0A VCC

CAS0A = CAS0 * COPRD                    ;coprocessor read access
       + CAS0 * /COPRD * /COPWR         ;local read access
       + CAS0 * COPWR * WB0             ;coprocessor write access

CAS0B = CAS0 * COPRD                    ;coprocessor read access
       + CAS0 * /COPRD * /COPWR         ;local read access
       + CAS0 * COPWR * WB1             ;coprocessor write access

CAS0C = CAS0 * COPRD                    ;coprocessor read access
       + CAS0 * /COPRD * /COPWR         ;local read access
       + CAS0 * COPWR * WB2             ;coprocessor write access

CAS0D = CAS0 * COPRD                    ;coprocessor read access
       + CAS0 * /COPRD * /COPWR         ;local read access
       + CAS0 * COPWR * WB3             ;coprocessor write access

CAS1A = CAS1 * COPRD                    ;coprocessor read access
       + CAS1 * /COPRD * /COPWR         ;local read access
       + CAS1 * COPWR * WB0             ;coprocessor write access

CAS1B = CAS1 * COPRD                    ;coprocessor read access
```

```
       + CAS1 * /COPRD * /COPWR          ;local read access
       + CAS1 * COPWR * WB1              ;coprocessor write access

CAS1C = CAS1 * COPRD                     ;coprocessor read access
       + CAS1 * /COPRD * /COPWR          ;local read access
       + CAS1 * COPWR * WB2              ;coprocessor write access

CAS1D = CAS1 * COPRD                     ;coprocessor read access
       + CAS1 * /COPRD * /COPWR          ;local read access
       + CAS1 * COPWR * WB3              ;coprocessor write access

DESCRIPTION:
Listing of U32 (PAL16L8) providing the
nibble access in the display memory.
```

Table 4.1 Status register of WD279X for I, II and III command groups.

Status for Type I commands

Bit number	Signal name	Description
B0	Busy	High when command in progress
B1	Index	High when index mark detected
B2	Track 0	High when head is set on track 0
B3	CRC error	CRC error in ID field
B4	Seek error	High when seeked track did not variy corrctly
B5	Head loaded	High when head is loaded and engaged
B6	Protected	High when disk is write protected
B7	Not ready	High when drive not ready

Status for Type I and II commands

Bit number	Signal name	Description
B0	Busy	High when command in progress
B1	Data request	High when ready for data transfer
B2	Lost data	High when data over-run or under-run error
B3	CRC error	If B4 is set CRC error in ID field If B4 is not set CRC error in data field
B4	Record Not Found (RNF)	When high, indicates that track, sector or side were not found
B5	Record type	Used for read record: 0 = data mark 1 = deleted data mark
B6	Write protected	High when disk is write protected
B7	Not ready	High when drive not ready

Table 4.2 Command summary and flag summary for WD279X faimly.

Commmand Type	Command	WD2791 WD2793						WD2795 WD2797					
		Bit Numbers						Bit Numbers					
		765	4	3	2	1	0	765	4	3	2	1	0
I	Restore	000	0	h	v	r1	r0	000	0	h	v	r1	r0
I	Seek	000	1	h	v	r1	r0	000	1	h	v	r1	r0
I	Step	001	t	h	v	r1	r0	001	t	h	v	r1	r0
I	Step-in	010	t	h	v	r1	r0	010	t	h	v	r1	r0
I	Step-out	011	t	h	v	r1	r0	011	t	h	v	r1	r0
II	Read Sector	100	m	s	e	c	0	100	m	l	e	u	0
II	Write Sector	101	m	s	e	c	a0	101	m	l	e	u	a0
III	Read Address	110	0	0	e	0	0	110	0	0	e	u	0
III	Read Track	111	0	0	e	0	0	111	0	0	e	u	0
III	Write Track	111	1	0	e	0	0	111	1	0	e	u	0
IV	Force Interrupt	110	1	i3	i2	i1	i1	110	1	i3	i2	i1	i0

h = Enable Head load at beginning
v = Enable Varify track number
t = Update track number
a0 = Deleted data address mark
c = Enable side compare
u = side output
e = 15 mS delay
s = compare side
m = Enable multiple sector operation
r0-r1 = step rate; 0 = 6.0 mS
1 = 12.0 mS
2 = 20.0 mS
3 = 30.0 mS
l = sector length flag
in = Interrupt condition flag
i0 = Interrupt on non-ready to ready transition
i1 = Interrupt on ready to non-ready transition
i2 = Interrupt on index pulse
i3 = Immediate Interrupt
i0-i3 = 0: Disable interrupt.

Table 4.3 Jumper's table for WD279X disk controller interface circuit.

Jumper	Function
J1	On : For calibration of RV1, RV2 and CV. Off: For smooth running
J2	On : 5.25 inch floppy disk drive selected Off: 8.00 inch floppy disk drive selected
J3	On : For 8.00 inch floppy drives (input clock = 2Mhz)
J4	On : For 5.25 inch floppy drives (input clock = 1Mhz)
J5	For WD2791 and WD2793 (not used for WD2795 and WD2797) On : Enable internal divider (2MHz input clock can be used for 5.25 inch floppy drives) Off: Disable internal divider (use 1Mhz for 5.25 inch floppy drives) (use 2Mhz for 8.00 inch floppy drives)
J6	For WD2795 and WD2797 (not used for WD2791 and WD2793) On : Side Select Output (SSO)
J7	For WD2795 and WD2797 (not used for WD2791 and WD2793) On : Side Select Output (SSO) (used along with J6)
J8	For WD2791 and WD2793 (not used for WD2795 and WD2797) On : SSO from the mask register Off: Single side floppy disk drives

Table 4.4 Status bytes from DP8474.

Status bytes 0, 1, 2 and 3 are read from the data register. The bit definition for the individual bytes is given below.

Status byte 0.

Bit number	Description
B0-B1	Drive selected (binary coded)
B2	Side at the end of execution
B3	Drive not ready
B4	Track 00 signal failed
B5	Seek or Recalibrate command completed
B6-B7	termination code 0 = normal termination 1 = Abnormal termination 2 = invalid command 3 = Ready changed state

Status byte 1.

Bit number	Description
B0	Missing address mark
B1	Write protected
B2	No data (address mark was found)
B3	Not used
B4	Over run error
B5	CRC error (in address or data field)
B6	Not used
B7	End of track found, therefore can not continue

Status byte 2.

Bit number	Description
B0	Missing address mark in data field
B1	Bad Track (track number is FFH)
B2	Scan did not match the condition
B3	Scan equal successful
B4	Wrong track number
B5	CRC error in data field
B6	Deleted data mark
B7	Not used

Status byte 3.

Bit number	Description
B0-B1	Drive selected (binary coded)
B2	Head/Side
B3	Not used
B4	Track 00 status
B5	Ready status
B6	write protect status
B7	Not used

Table 5.1 Status byte returned by the SCSI device.

Bit number	Description
B0	Reserved
B1-B4	Status code 0 = Good status 1 = Check condition 4 = Busy 8 = Intermediate good (for linked commands) C = Reservation conflict
B5	Reserved
B6	Reserved
B7	Reserved

Table 5.2 Status register on the SASI interface.

Bit number	Signal name	Description
B0	I/O	0 = Transfer from target 1 = Transfer to target
B1	C/D	0 = Command phase 1 = Data phase
B2	MSG	0 = Message phase 1 = No message phase
B3	BSY	0 = Device busy 1 = Device idle
B4	REQ	0 = Transfer request from device 1 = Device not ready for data transfer

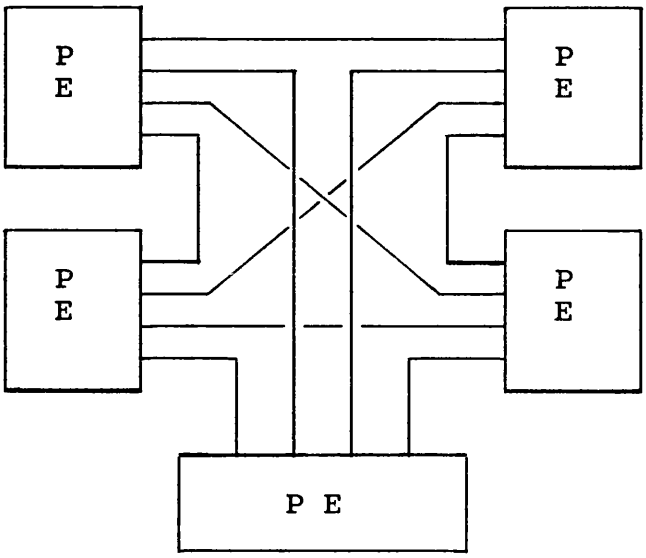
Table 5.3 Mask register (SASI interface).

Bits Number	Condition Code Code
B0 (LSB)	1 Assert SEL signal 0 Remove SEL signal
B1	1 Assert 'reset' 0 Remove 'reset'
B2	1 Enable DMA requests 0 Disable DMA requests
B3-B4	0 No interrupts during command, data or result phases 1 Interrupt during data phase No interrupt for command or result phases 2 No interrupts during data phase but interrupt during command or result phases 3 Interrupt in all, data, command or result phases
B5	1 Do not interrupt on message phase 0 Interrupt on message phase
B6	1 Enable interrupt 0 Disable interrupt

N.B. D6 effectively masks D3, D4 and D5.

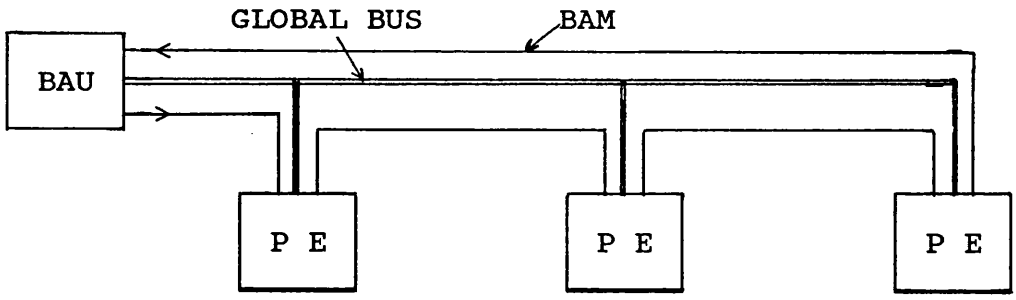
Table 5.4 Pin assognment for Centronics bus.

Pin number	Name of signal	Signal directoion	Functional description
1	Strobe	Input	Pulse after data settled
2	D0 (LSB)	Input	Data bit 0
3	D1	Input	Data bit 1
4	D2	Input	Data bit 2
5	D3	Input	Data bit 3
6	D4	Input	Data bit 4
7	D5	Input	Data bit 5
8	D6	Input	Data bit 6
9	D7	Input	Data bit 7
10	ACK	Output	Pulse when ready
11	Busy	Output	High when busy
12	PO	Output	High when 'Paper Out'
13	Select	Output	High when on-line
14	0V		Zero volts
15	N/C		
16	0V		Zero volts
17	GND		Chasis ground
18	+V	Output	+5.0 volts
19-30	GND		Signal ground
31	Prime	Input	Pulse to initialise printer
32	Error	Output	High when any error occurs
33	GND		Signal ground
34	N/C		
35	N/C		
36	N/C		



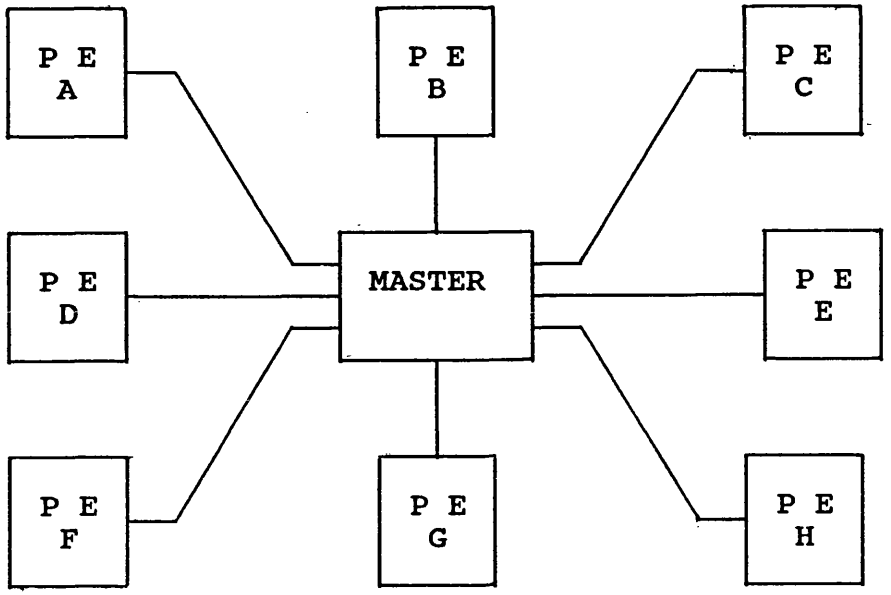
PE = Processing Element

Fig 2.1 A fully interconnected multi-computer system.



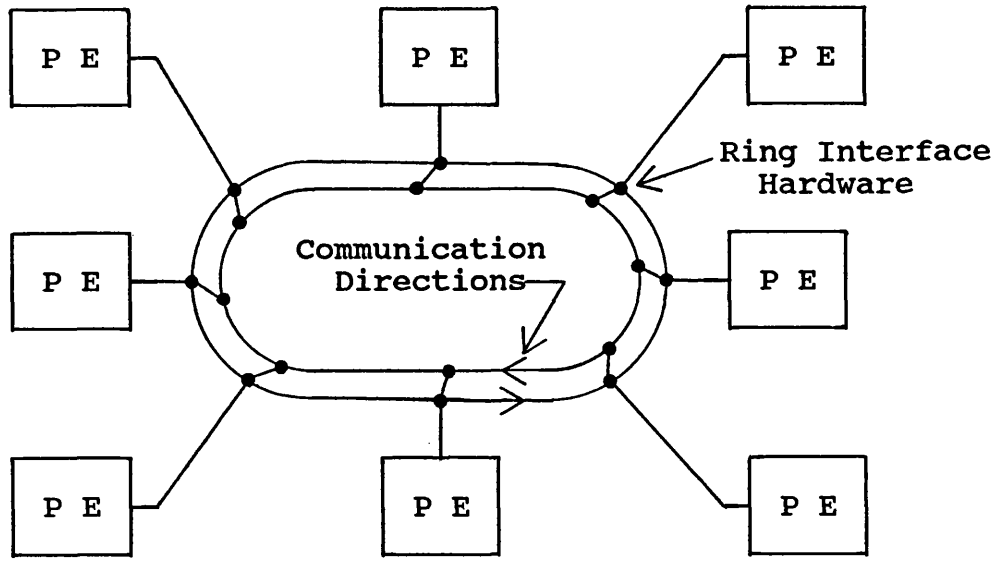
BAU = Bus Arbitor Unit
PE = Processing Element
BAM = Bus Arbiteration Mechanism

Fig 2.2 A shared bus architecture with single bus.



PE = Processing Element

Fig 2.3 Star configuration.



PE = Processing Element

Fig 2.4 Ring configuration.

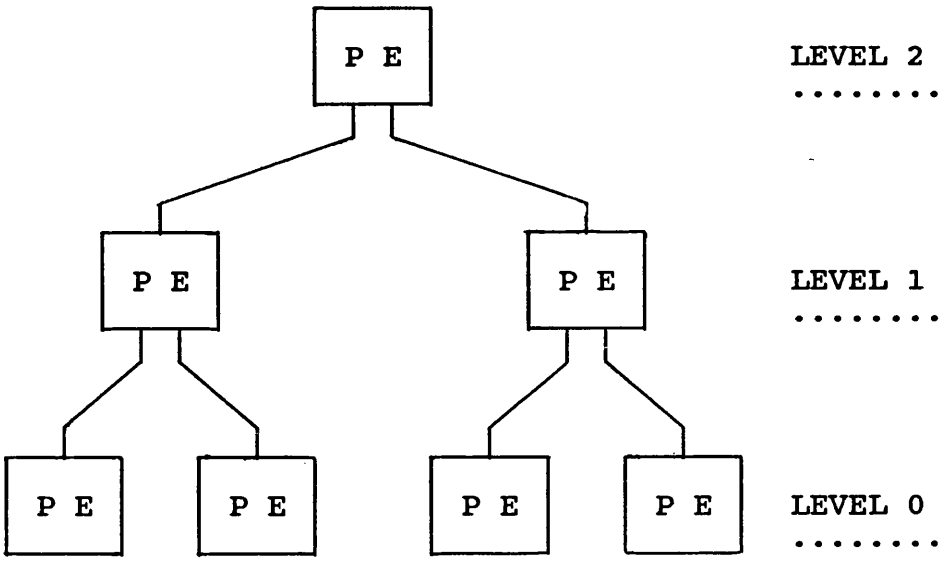


Fig 2.5 Three level tree architecture.

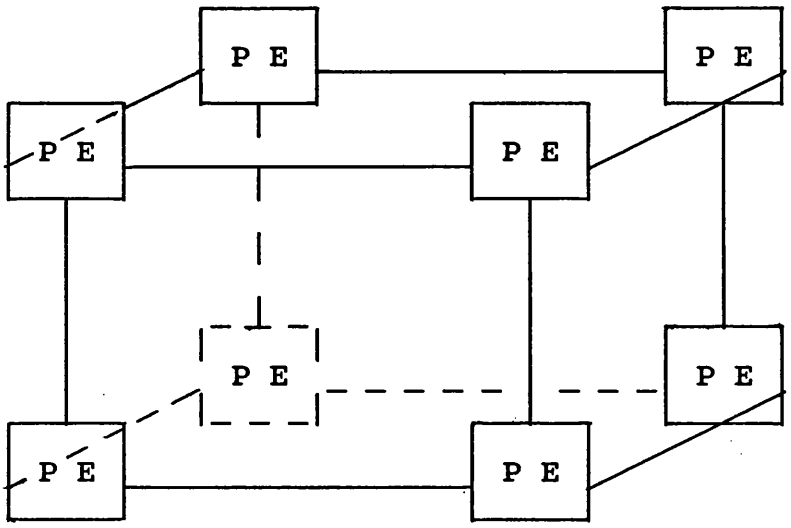
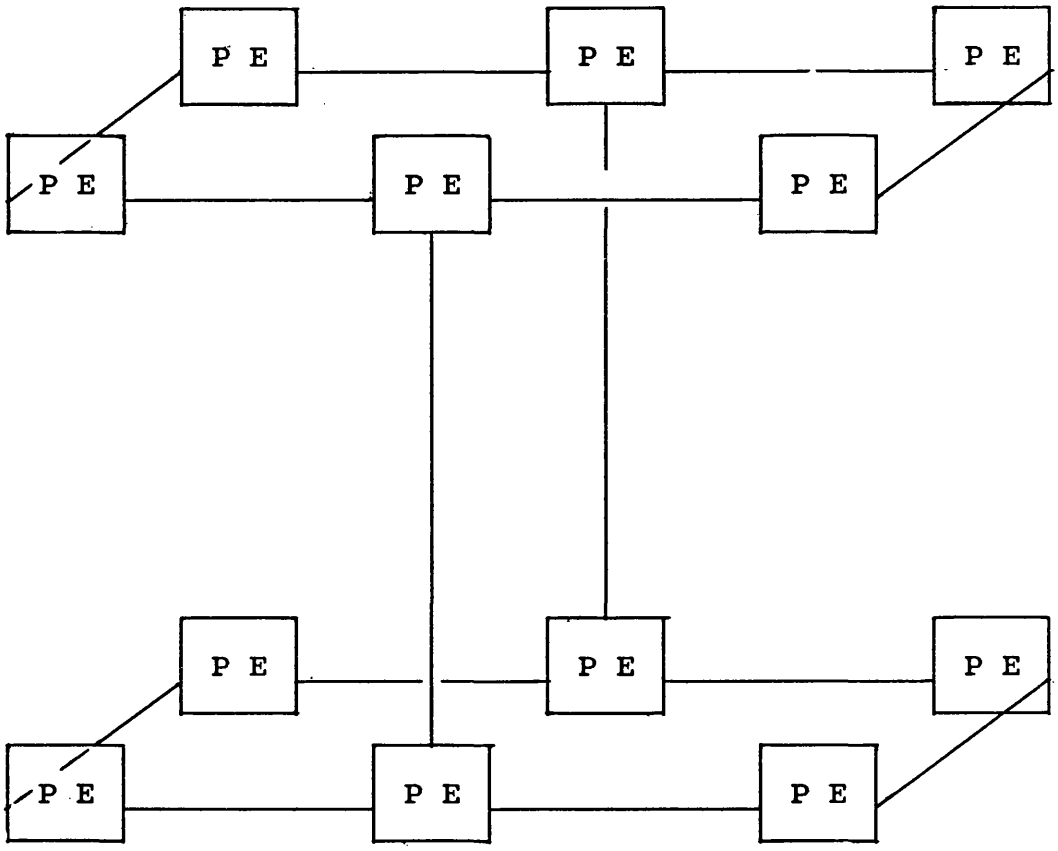
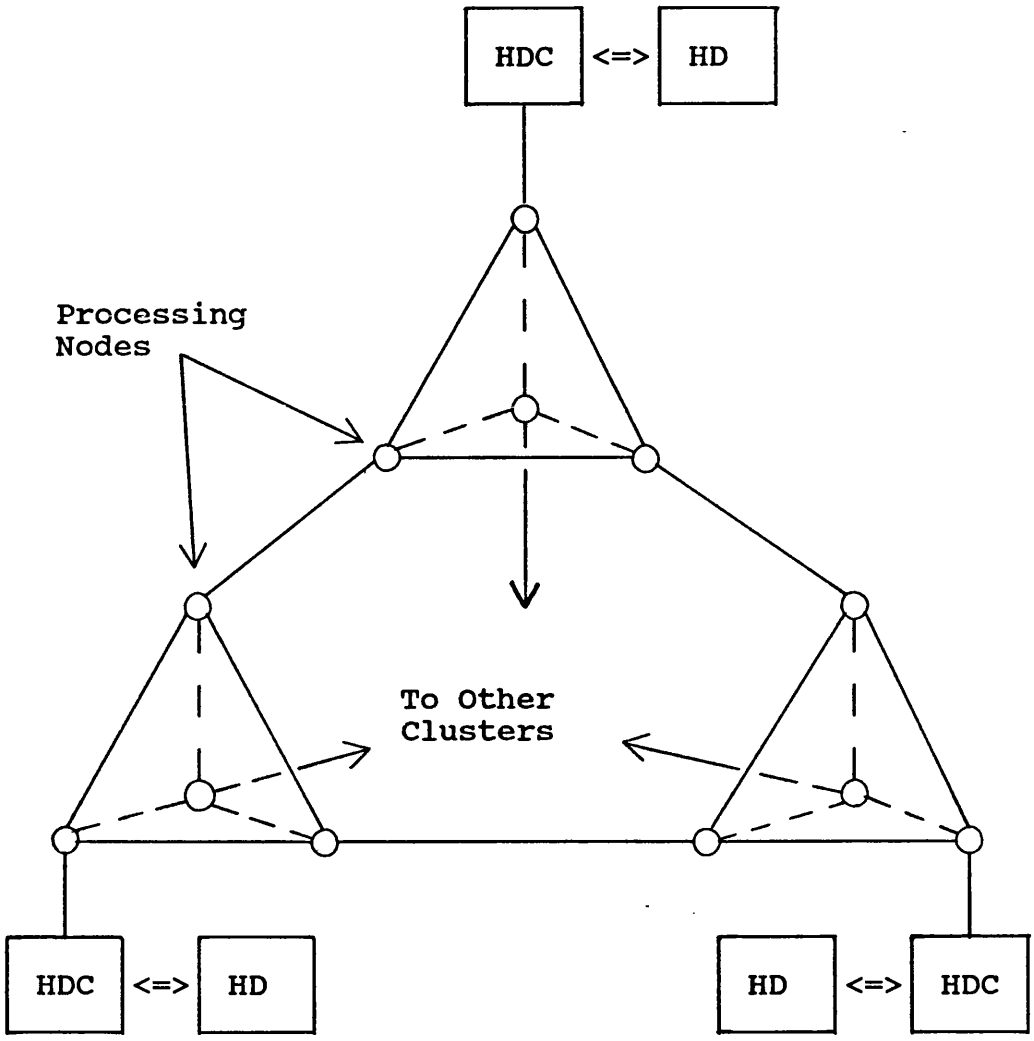


Fig 2.6 A normal 3-dimentional hyper cube with 8 processing elements.



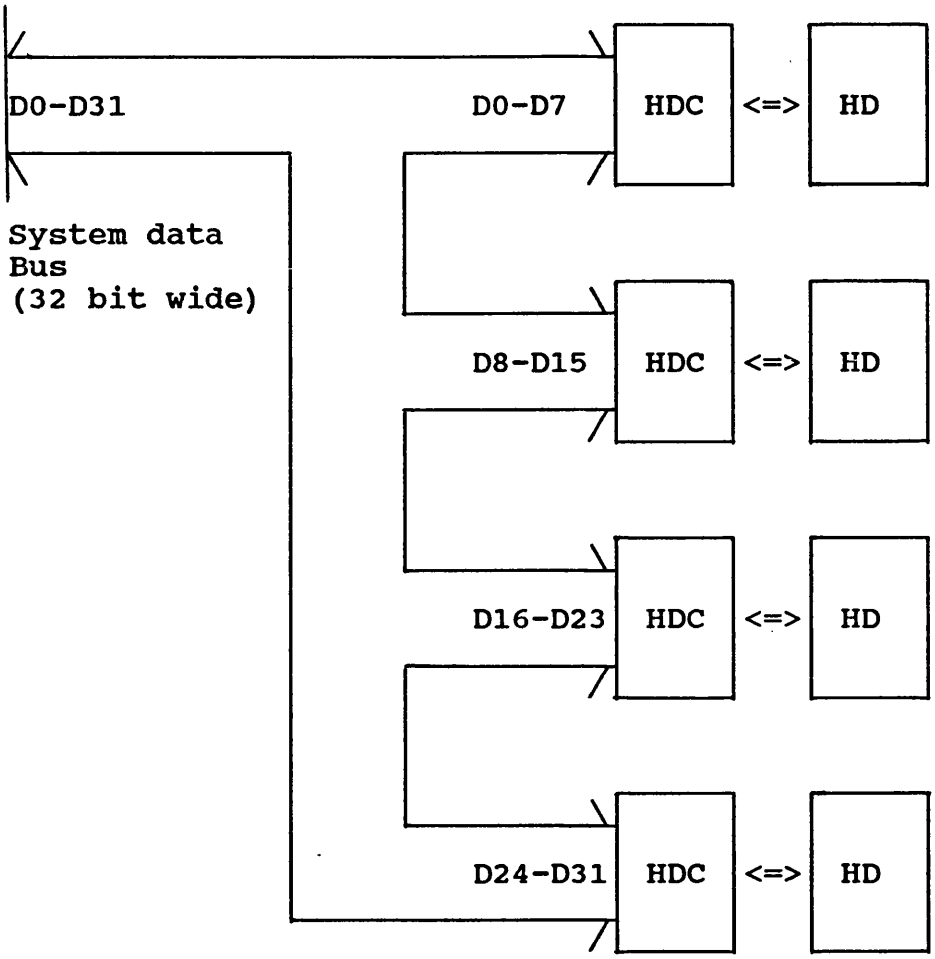
PE = Processing Element

Fig 2.7 A double bus hyper cube.



HD = Hard Disk unit
HDC = Hard Disk Controller card

Fig 2.8 Allocation of hard disk units to clusters of processing nodes.



HD = Hard Disk unit
HDC = Hard Disk Controller card

Fig 2.9 A possible Combined Disk Arrangement (CDA). 32 bit system data is stored on four different hard disks, 8 bit each, in parallel.

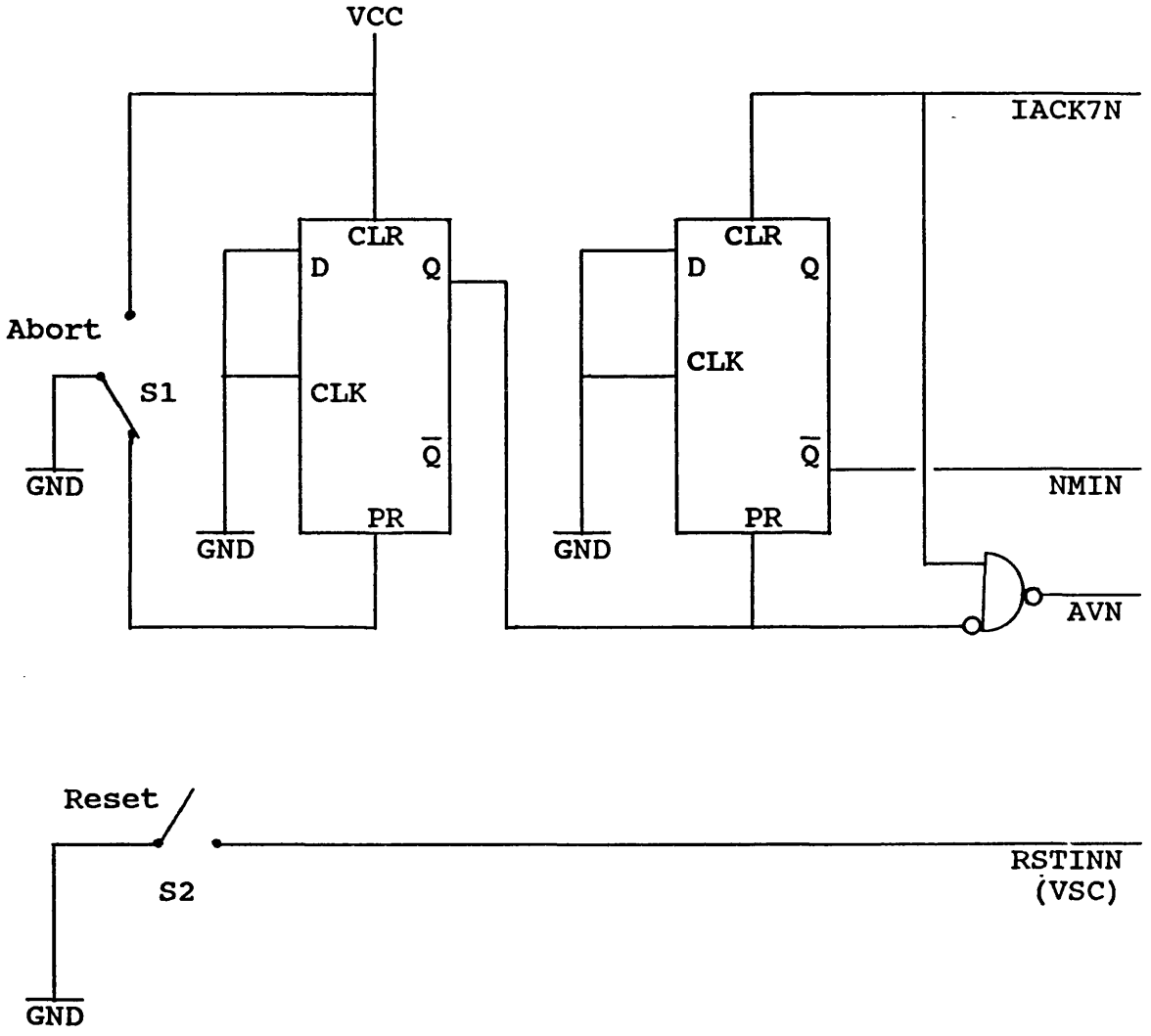


Fig 3.1 Reset and Abort circuits for microcore.

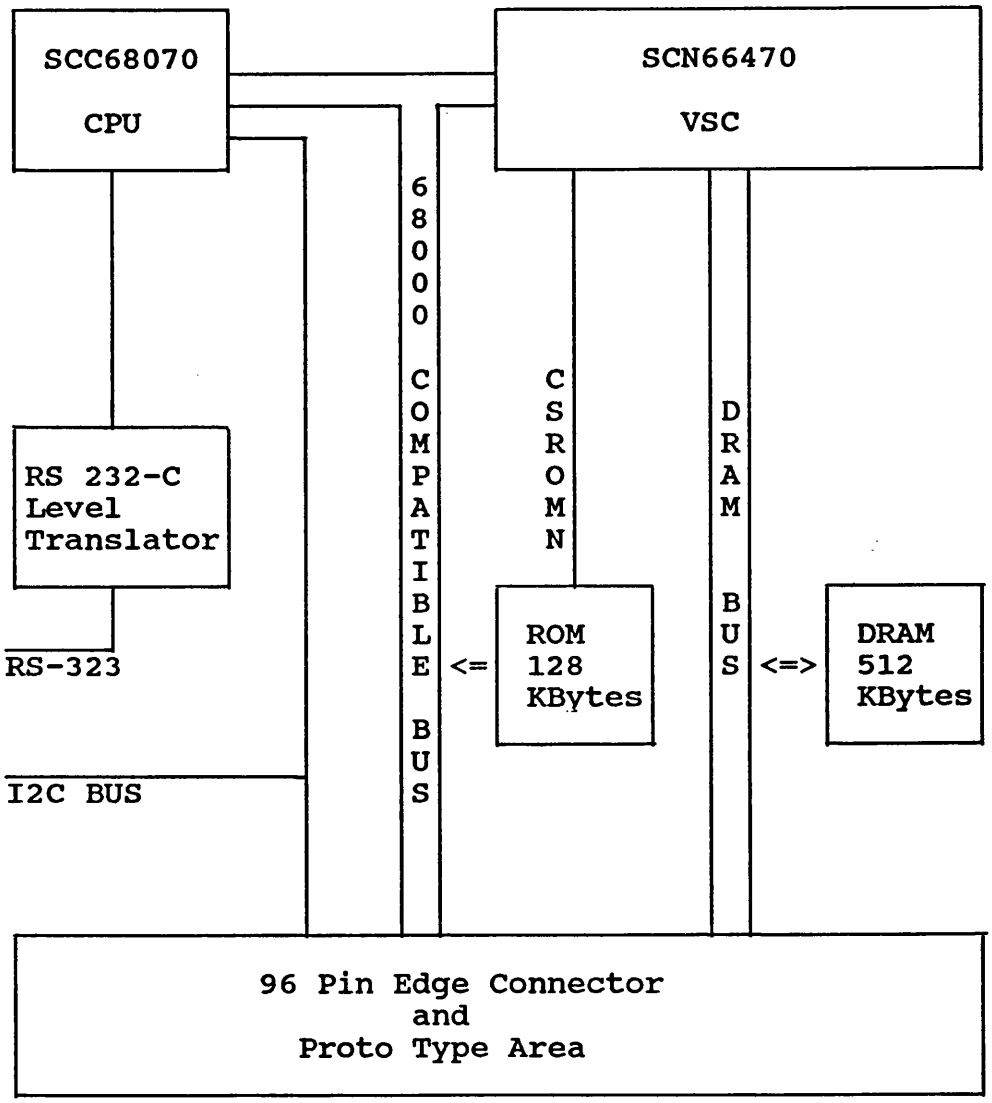


Fig 3.2 Micro Core Block Diagram.

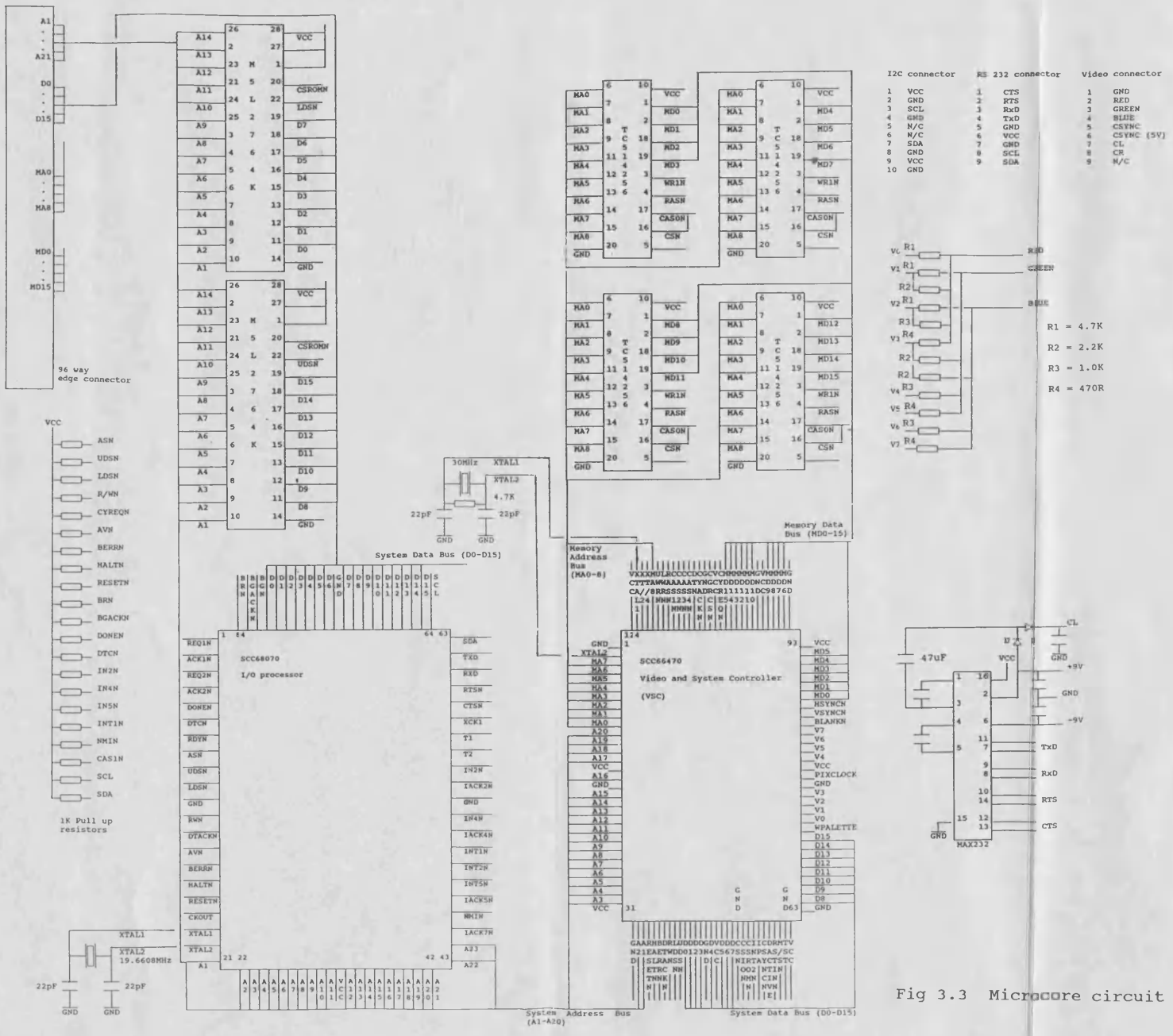


Fig 3.3 Microcore circuit diagram.

Pin number	row a	row b	row c
1	D12	D3	D4
2	D13	D2	D5
3	D11	D1	D6
4	D14	D0	D7
5	D15	A1	A2
6	D8	A11	A3
7	D9	LDSN	UDSN
8	D10	A4	A12
9	GND	NC	GND
10	A5	A10	A9
11	A6	A7	A14
12	RDYN	NC	A8
13	A13	DTCN	DONEN
14	T1	ACK2N	REQ2N
15	T2	ACK1N	REQ1N
16	NC	SDA	IACK7N
17	NC	SCL	NMIN
18	XT/2	INT1N	RESETN
19	RASN	DTACKN	RWN
20	NC	CSION	ASN
21	NC	CYREQN	CYACKN
22	NC	MA3	MA4
23	NC	MA2	MA5
24	MA1	MA7	MA6
25	MA0	MD6	MA8
26	MD8	MD5	MD2
27	MD10	MD7	MD9
28	MD1	MD13	MD11
29	MD0	MD14	MD12
30	MD3	CAS2N	MD15
31	MD4	CAS4N	CAS3N
32	VCC	VCC	VCC

Fig 3.4 Pinning list of the euro-connector on microcore.

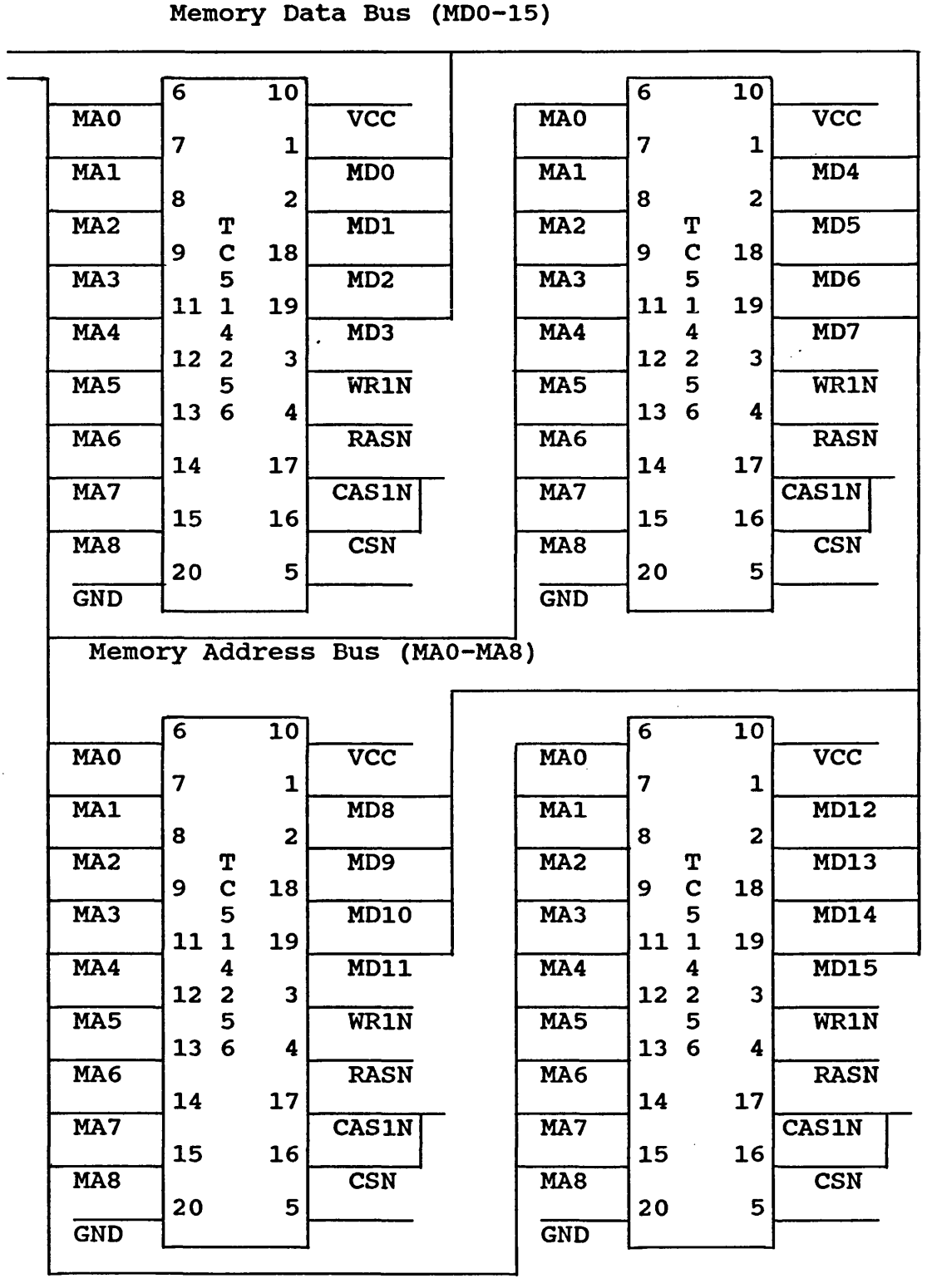


Fig 3.5 Memory extension on microcore.

Bit number	ISR register	DSCR register
B0	Not used	0 = 8 bit mode 1 = 16 bit mode
B1	Not used	Not used
B2	Not used	Not used
B3	Not used	Not used
B4	Interrupt from drive 0	Drive 0 busy
B5	Interrupt from drive 1	Drive 1 busy
B6	Interrupt from drive 2	Drive 2 busy
B7	Interrupt from drive 3	Drive 3 busy

Fig 4.1 Interrupt Source Register (ISR) and Device Control ans Status Register (DSCR) in the IMDC.

Byte address	Bit number 15 – 8	Bit number 7 – 0	Byte address
01	Command code	Main status	00
03	Extended status		02
05	Max retries	Actual retries	04
07	DMA count	Com. options	06
09	Buffer address MS byte		08
0B	Buffer address LS byte		0A
0D	Requested transfer length		0C
0F	Bytes transfered		0E
11	Cylinder number to be accessed		10
13	Head number	Sector number	12
15	Present cylinder		14
17	PRP Command control word.		16
19	SCWT pointer MS word		18
1B	SCWT pointer LS word		1A
1D	Scan terminator	Reserved	1C
1F	Max record length-1		1E
21	Pre index gap	Post index gap	20
23	Sync byte count	Post ID gap	22
25	Post data gap	Address mark count	24
27	Reserved		26
29-2D	ECC mask		28-2C
2F	Motor on delay.	Number of heads	2E
31	Ending sector	Step rate	30
33	Head settle time	Head load time	32
35	Seek time	Phase count	34
37	Low current boundry track number		36
3B-3F	ECC remainder		3A-3E
41	Number of cylinder per surface		40
43	Sector length	Flag byte	42
45-47	B-tree pointer		44-46
49-5B	IMDC working work area		48-5A

Fig 4.2 ECA block in the memory.

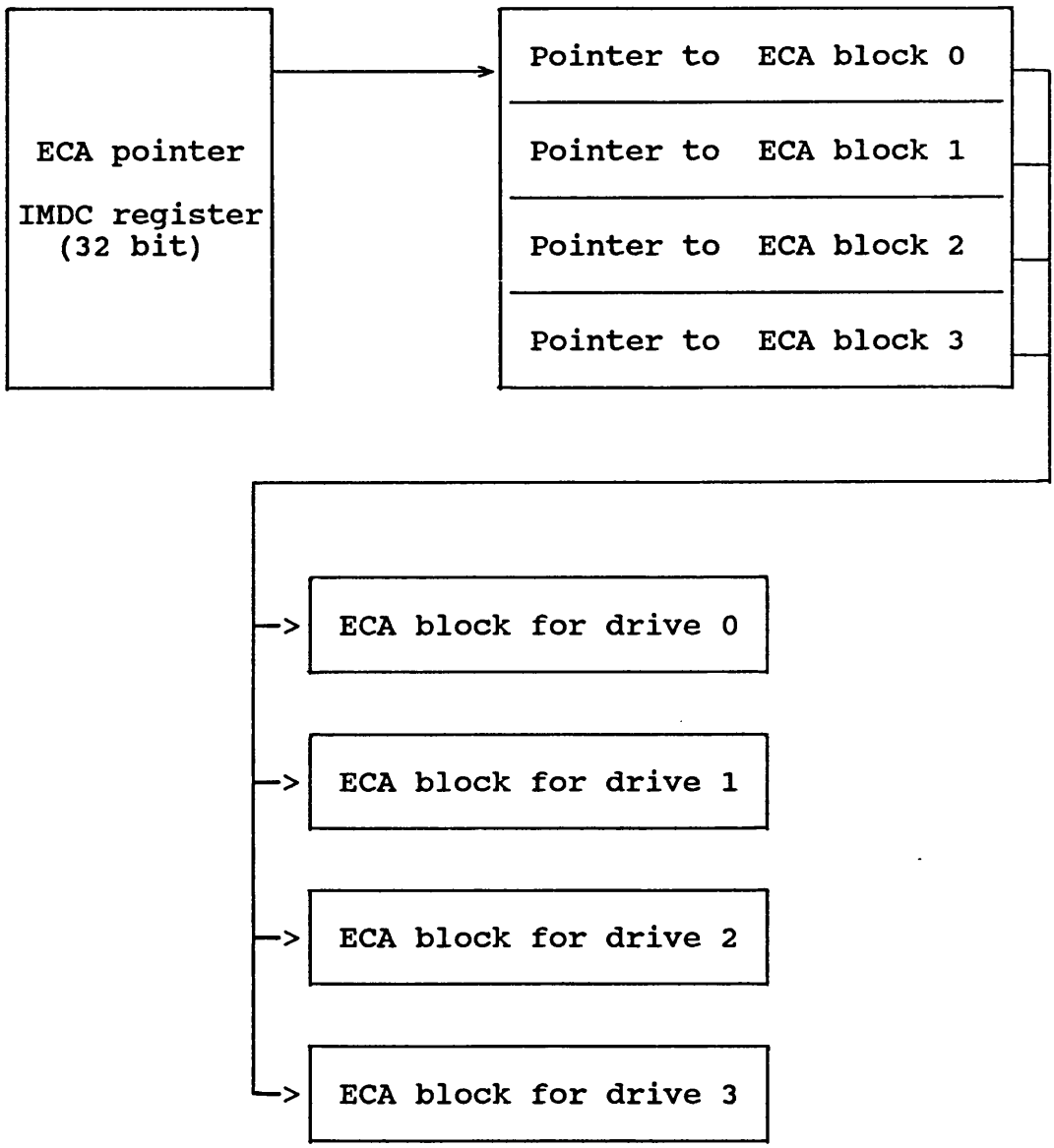


Fig 4.3 ECA pointers table in the memory.

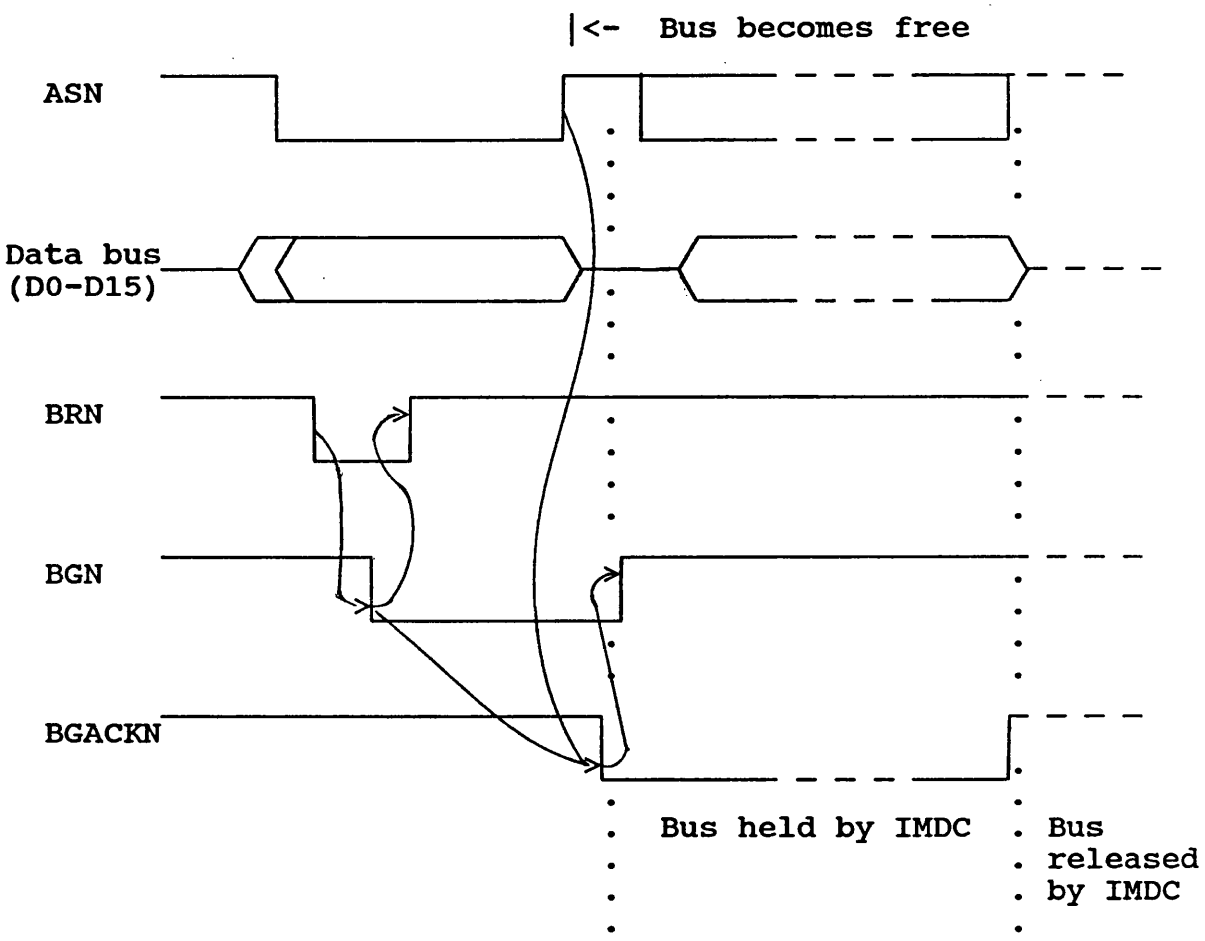


Fig 4.4 IMDC acquiring the bus for data transfers.

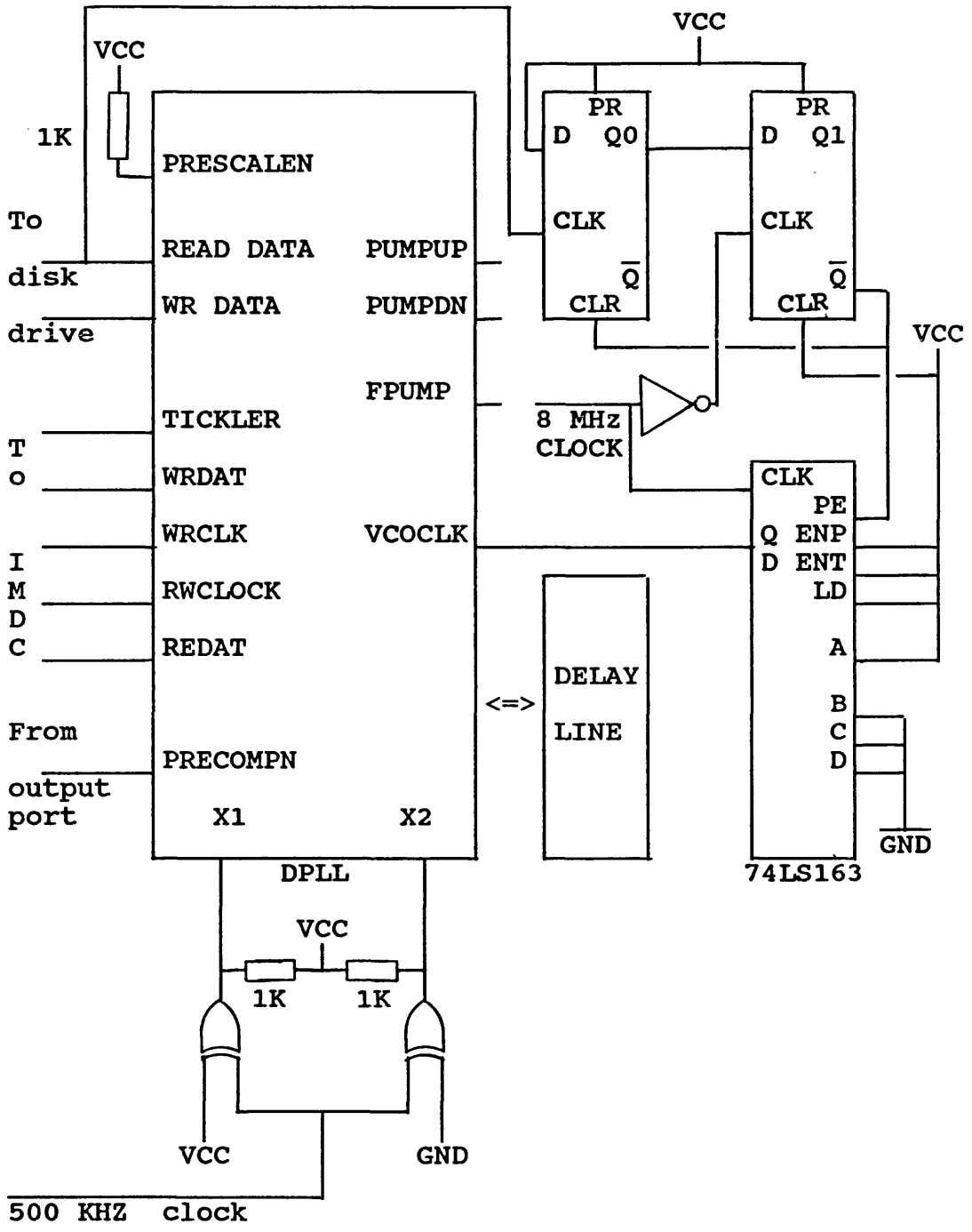


Fig 4.5 Flip-Flop arrangement to replace VCO.

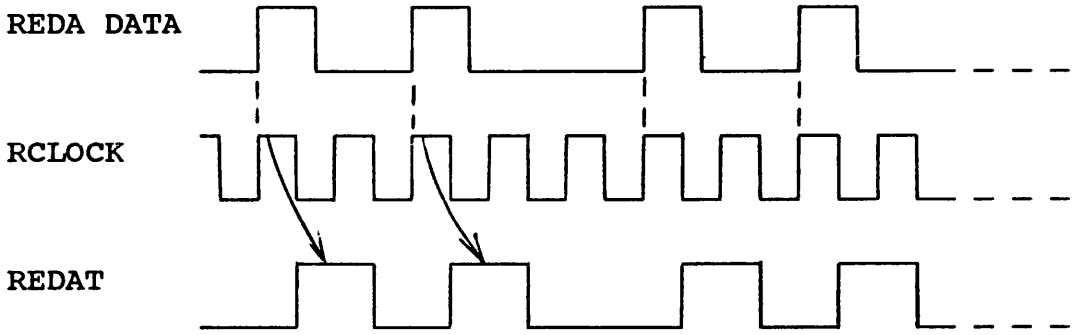


Fig 4.6 REDAT synchronised with the READ DATA and RCLOCK signals.

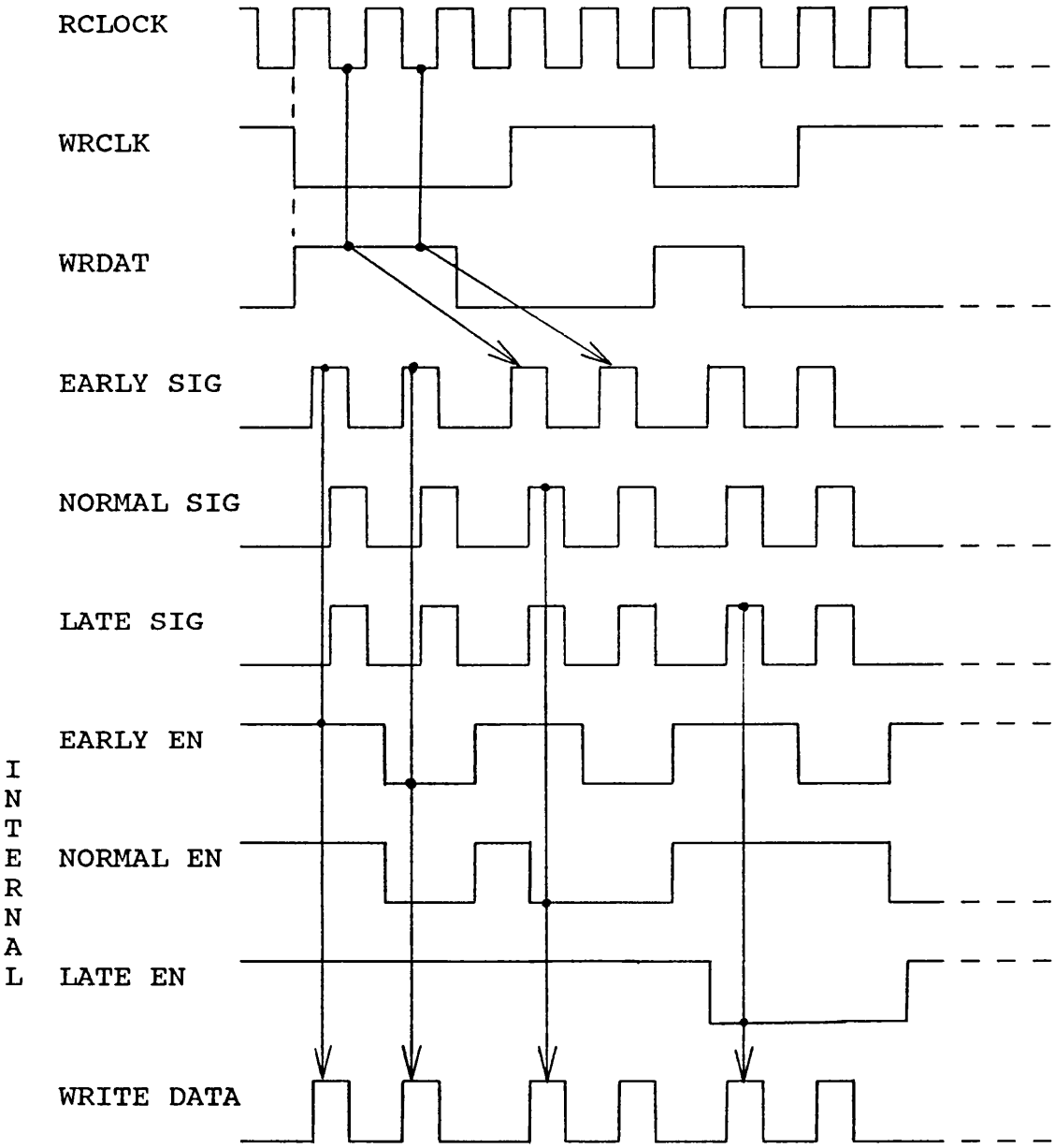


Fig 4.7 WRITE DATA signal synchronised with WCLOCK and EARLY, NORMAL and LATE signals.

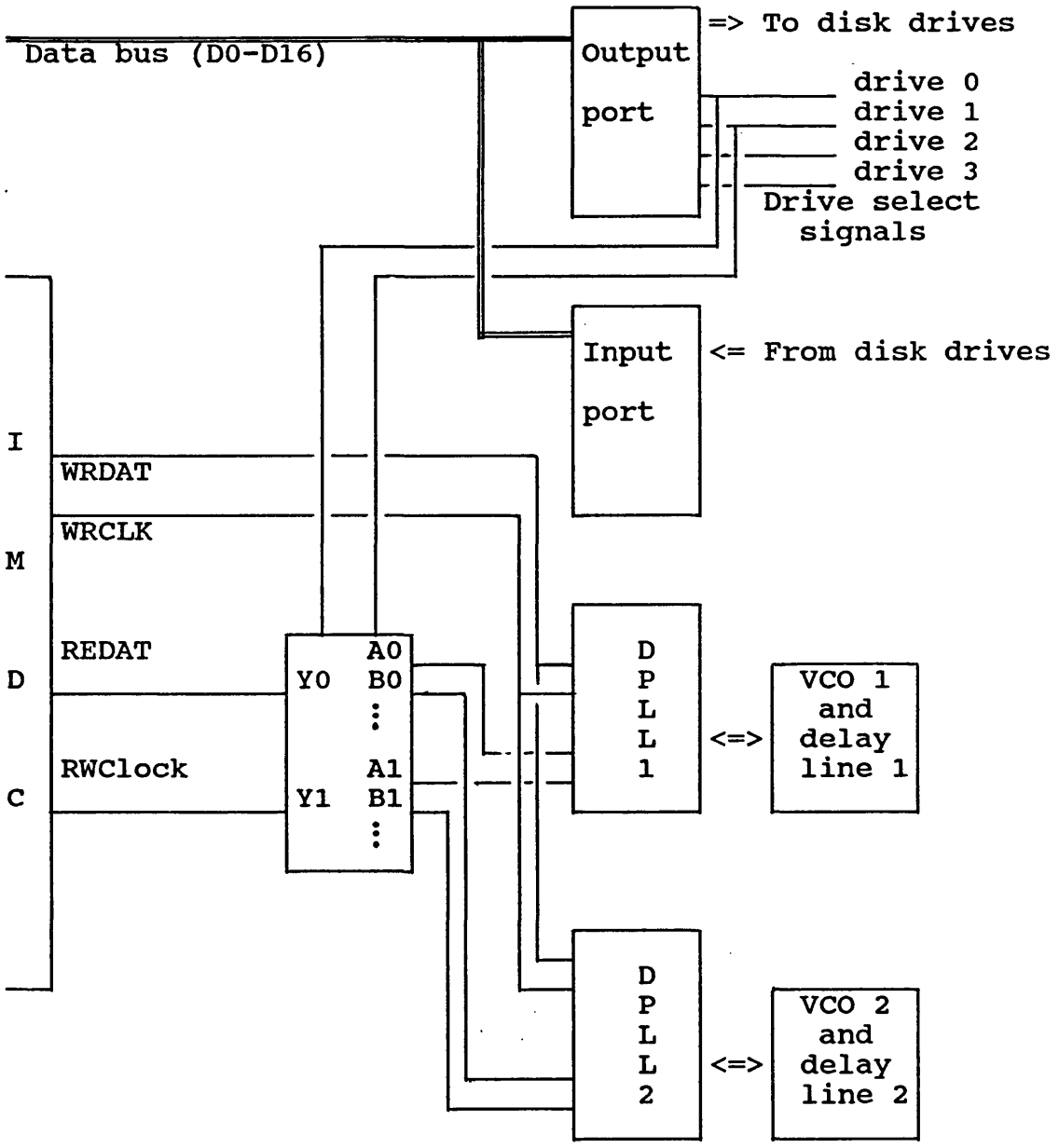


Fig 4.8 Decoder arrangement for a dual DPLL system.

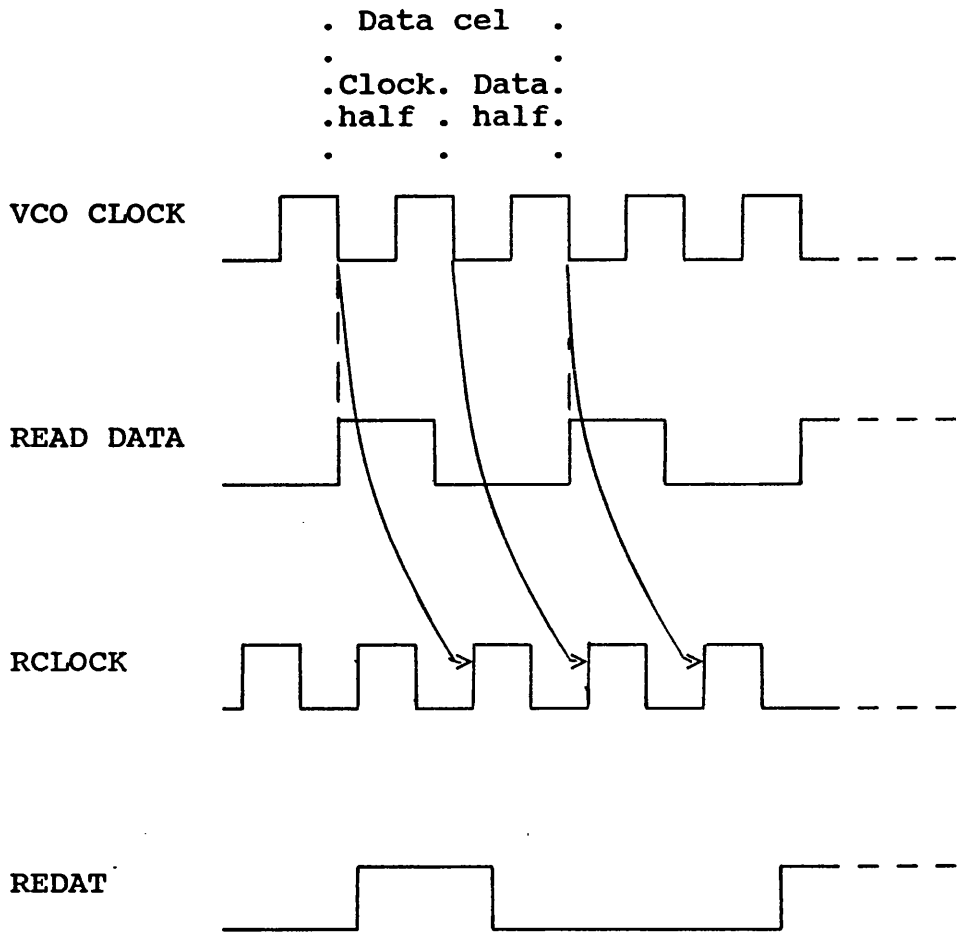


Fig 4.9 Synchronisation of 'READ DATA' and 'VCO clock' in MFM format.

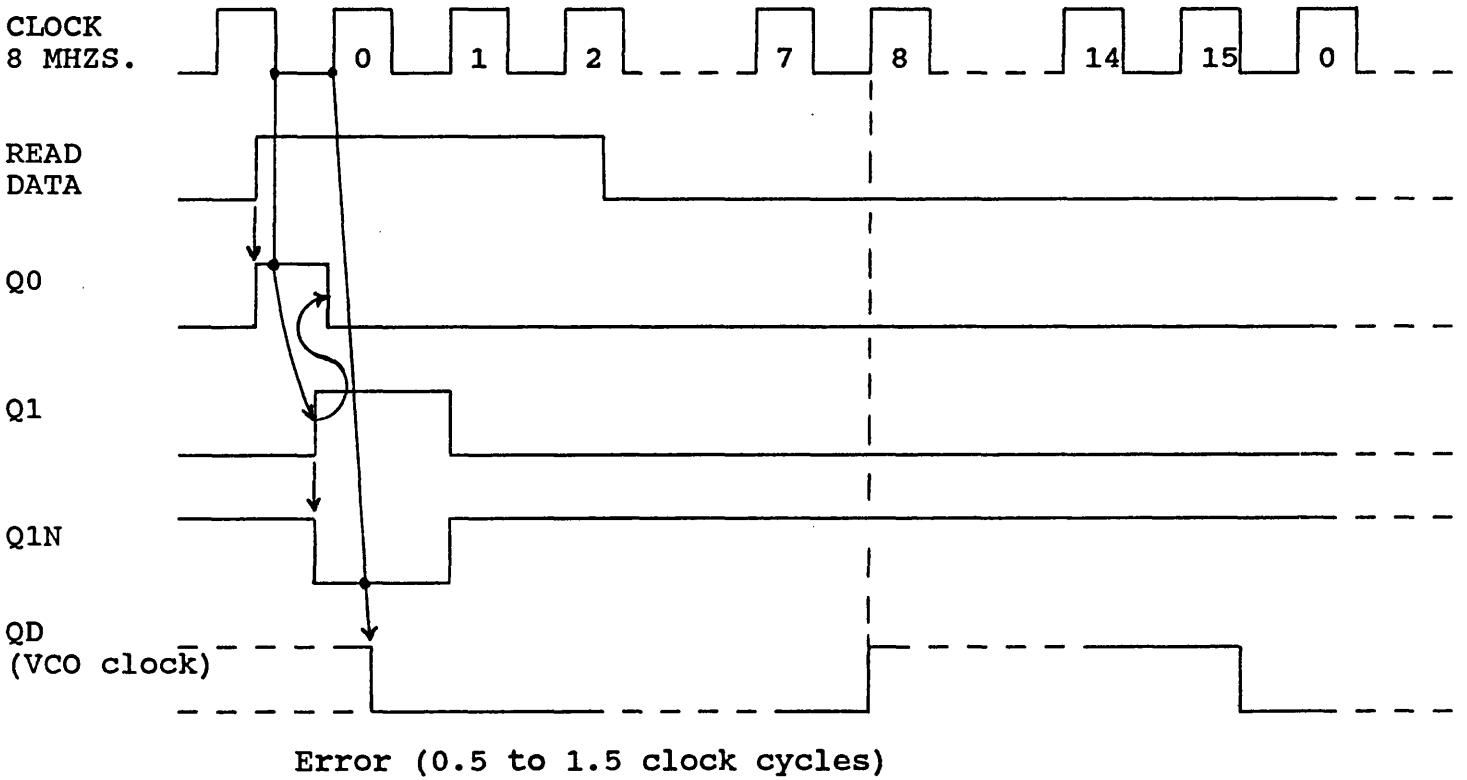


Fig 4.10 Synchronisation of 'VCO clock' with 'READ DATA' pulses in flip-flop arrangement.

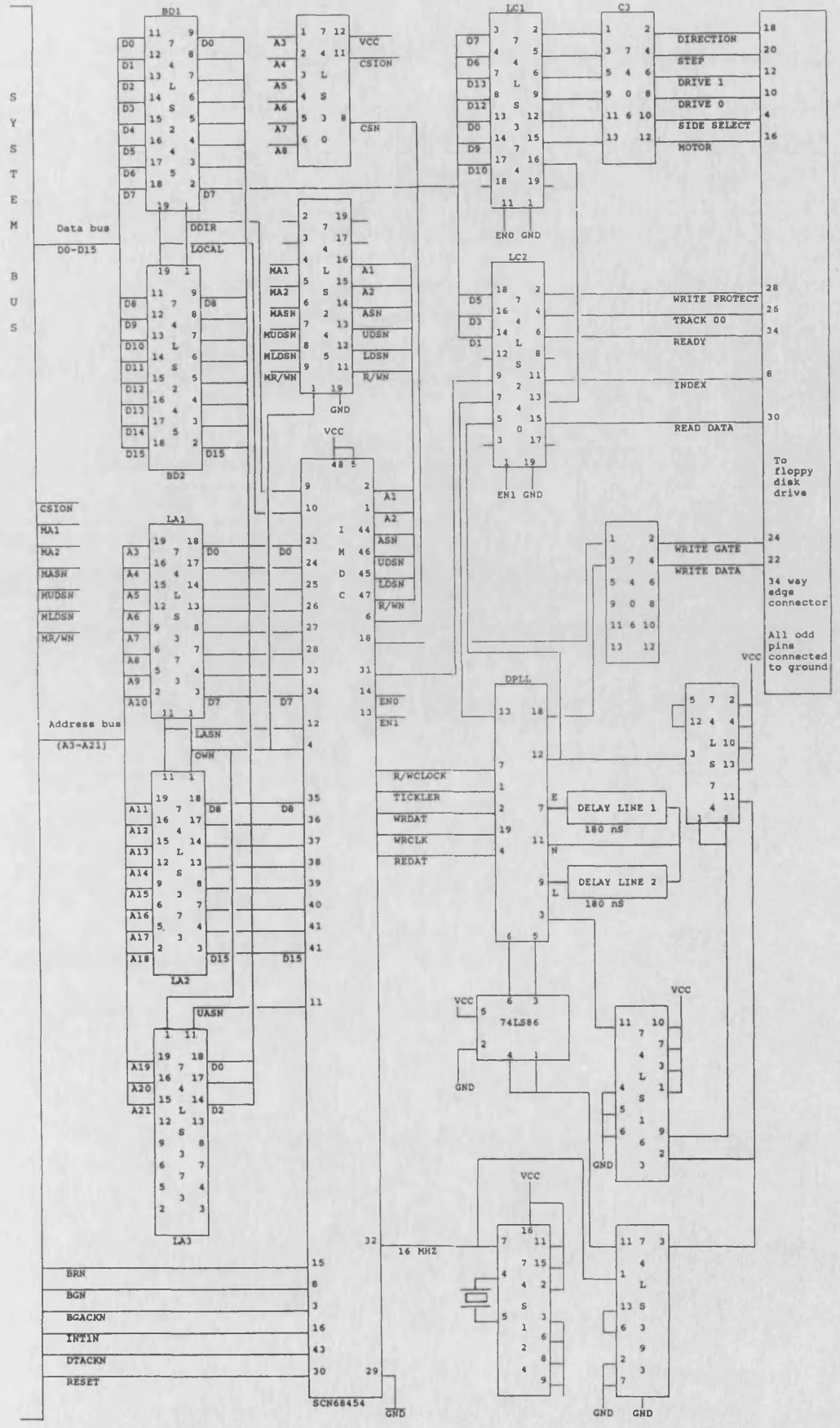


Fig 4.11 Circuit diagram for floppy disk controller interface circuit using SCN68454 (IMDC).

Input port definations.

Data bus line	Signal name
D0	Index
D1	Ready
D2	Seek complete
D3	Track 00 (first signal)
D4	Track 00 (second signal)
D5	Write protected (floppy)
D6	Write fault
D8-D15	Not used

Output port defination.

Data bus line	Signal name
D0	Head0*
D1	Head1*
D2	Head2*
D3	Head3*
D4	Head4*
D5	Not used
D6	Step pulse
D7	Direction
D8	Low write current
D9	Motor on
D10	Precompensation on
D11	Head load (floppy)
D12	Drive 0
D13	Drive 1
D14	Drive 2
D15	Drive 3

* Head0-4: Five bit binary coded head select signals. A total of 32 head drives can be used.

Fig 4.12 Input and Output port definations for IMDC.

Bit number	Function
B0 (LSB)	0 Enable single density format (FM) 1 Enable double density format (MFM)
B1	0 Disable interrupts 1 Enable interrupts
B2	0 Disable 'DRQ' 1 Enable 'DRQ'
B3	1 Select drive 0
B4	1 Select drive 1
B5	1 Select drive 2
B6	1 Select drive 3
B7	0 Select side 0 1 Select side 1

N.B. B7 is used with WD2791 and WD2793 floppy controllers only.

Fig 4.13 Bit description of the mask register added to the WD279X devices to implement the full FDC interface.

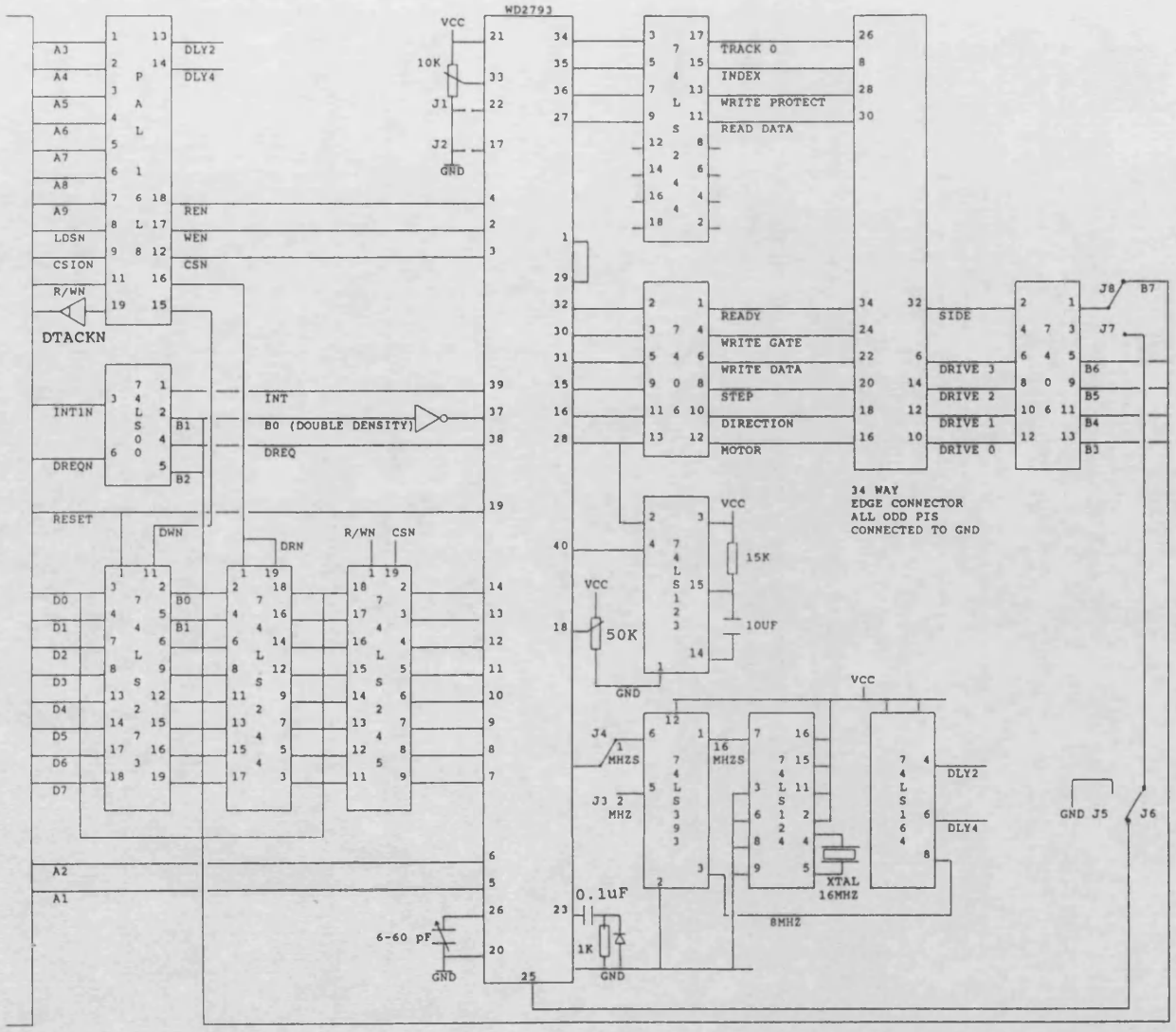


Fig 4.14 Circuit diagram for floppy disk controller interface using WD2793.

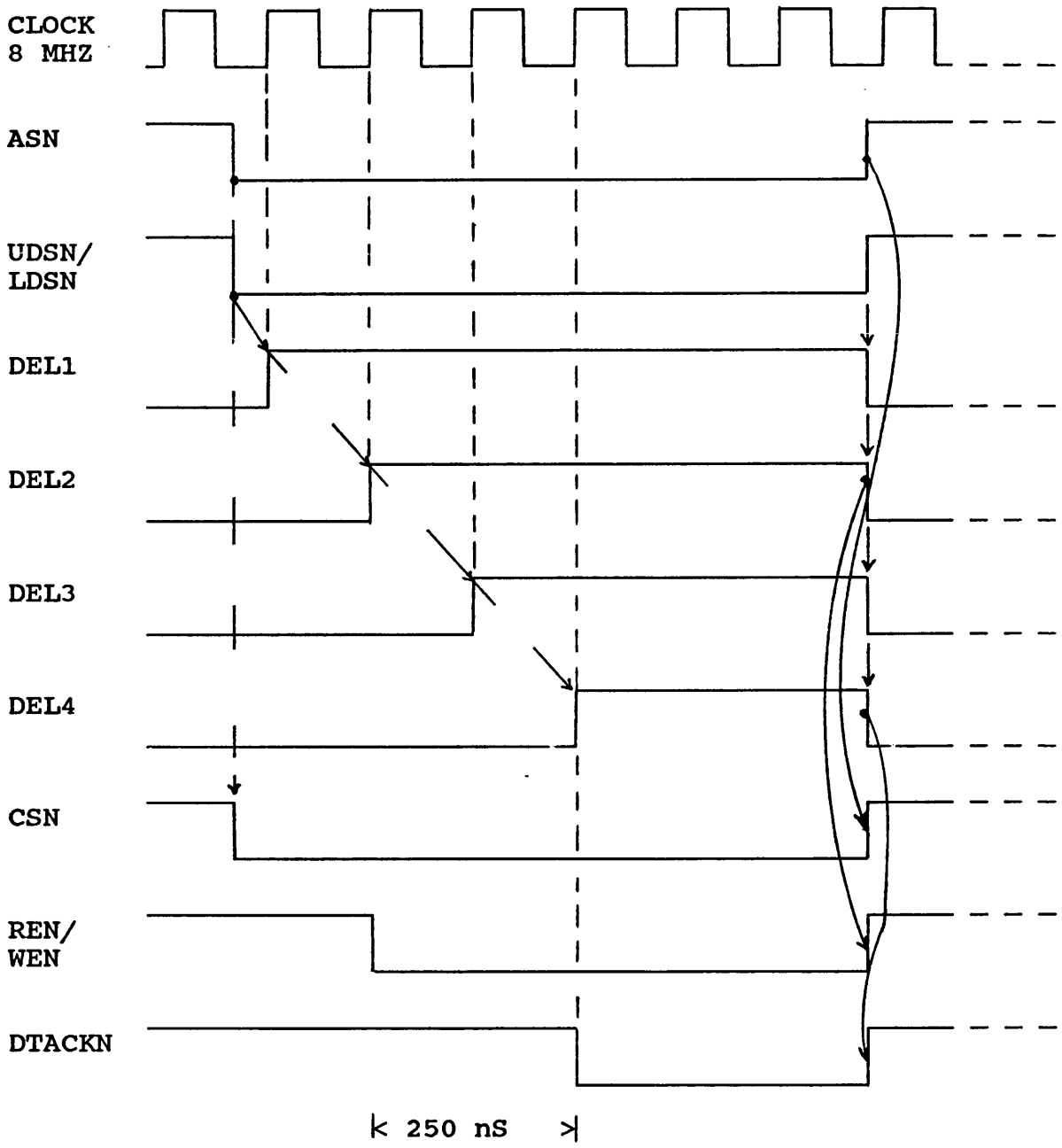


Fig 4.15 DTACKN and REN/WEN synchronised with their respective delay signals.

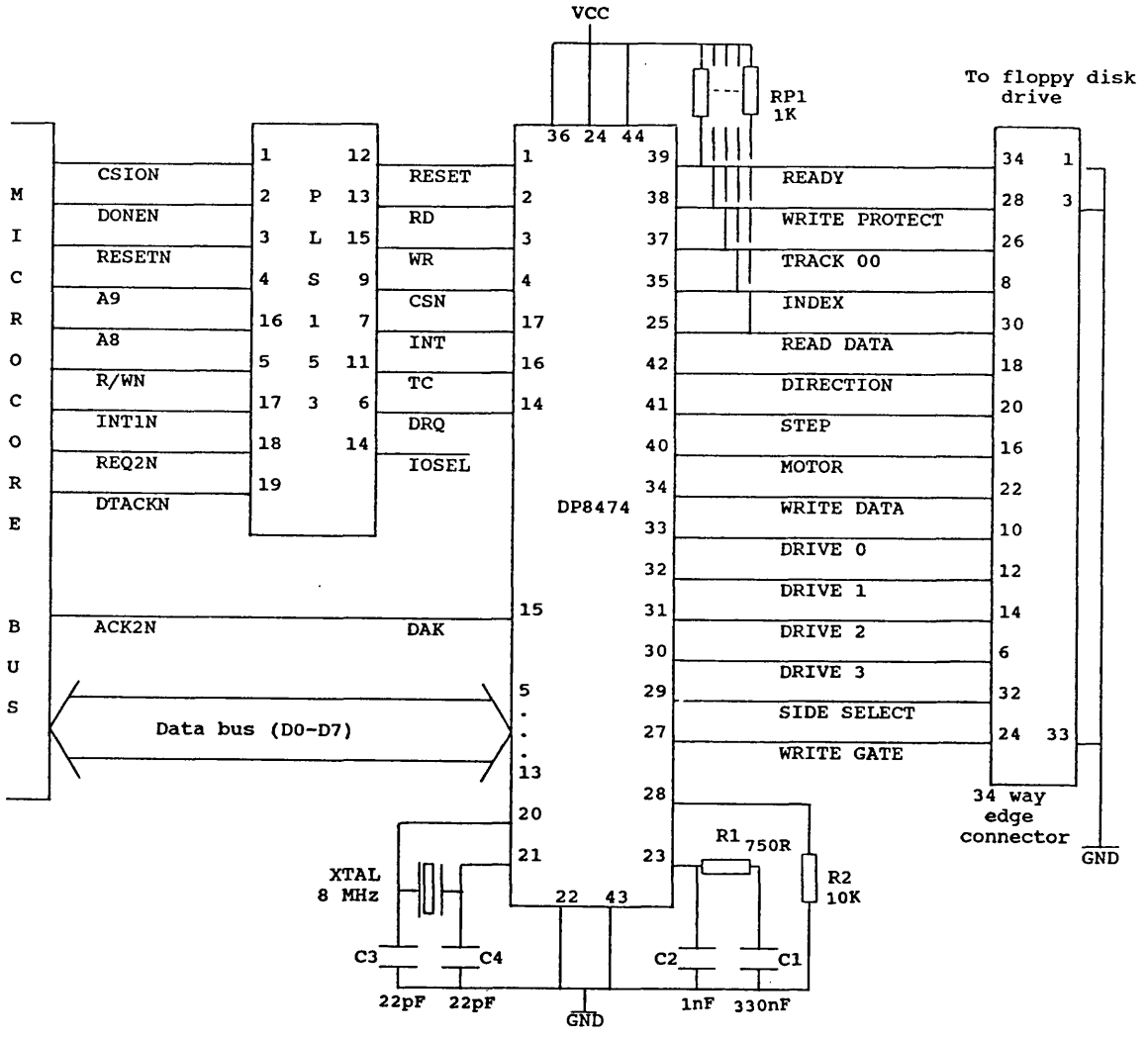


Fig 4.16 Circuit diagram for floppy disk controller interface circuit using 'DP8474'.

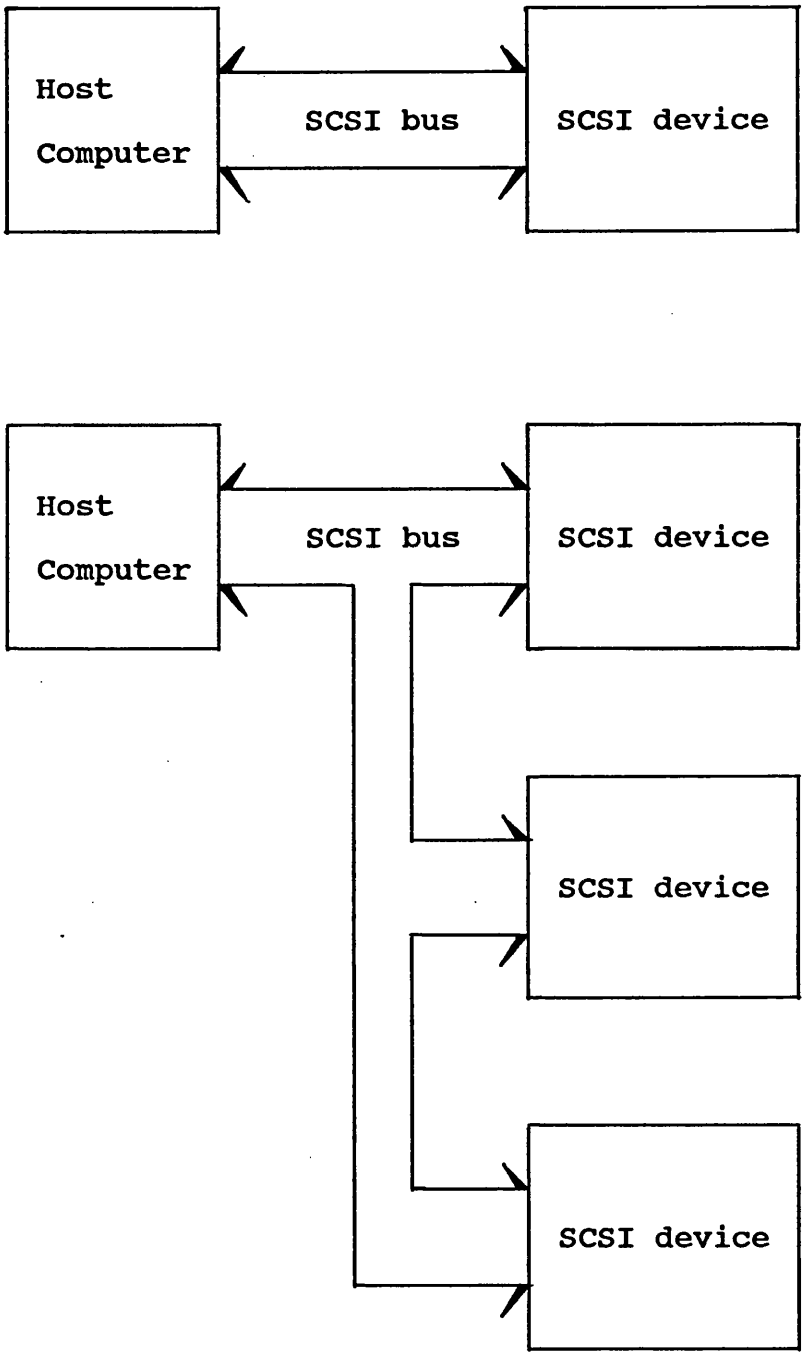


Fig 5.1 SCSI configuration. Single-initiator single-target and single-initiator multiple-target systems.

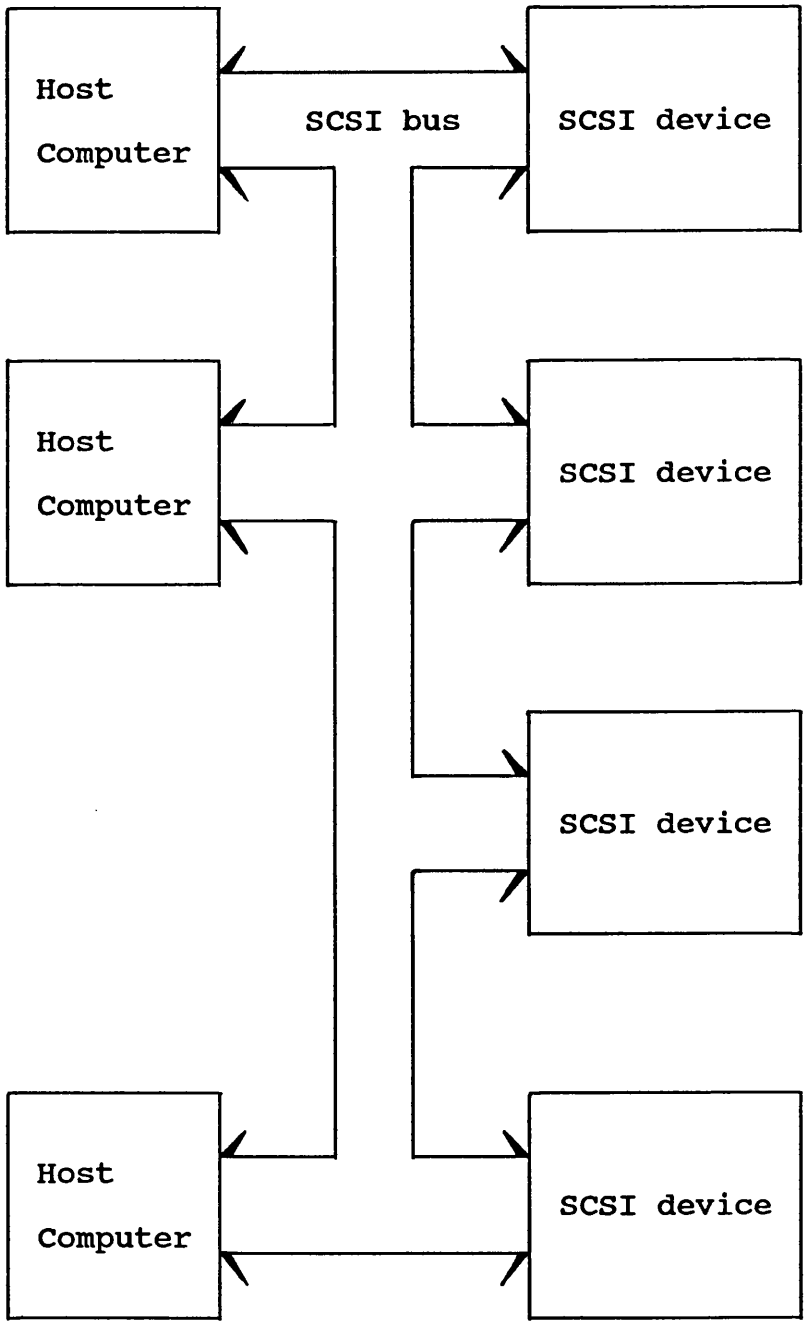
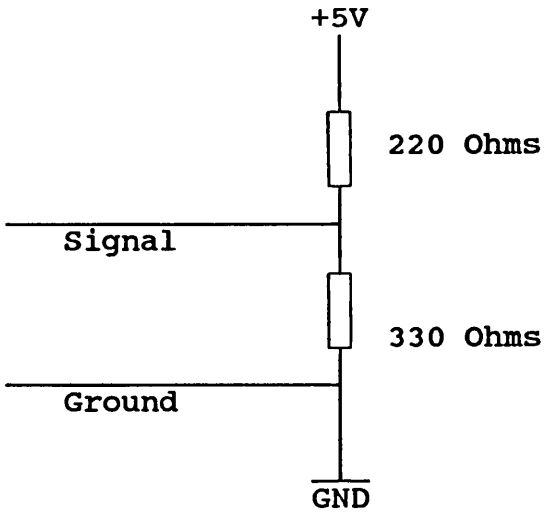


Fig 5.2 SCSI configuration. Multiple-initiator multiple-target system.

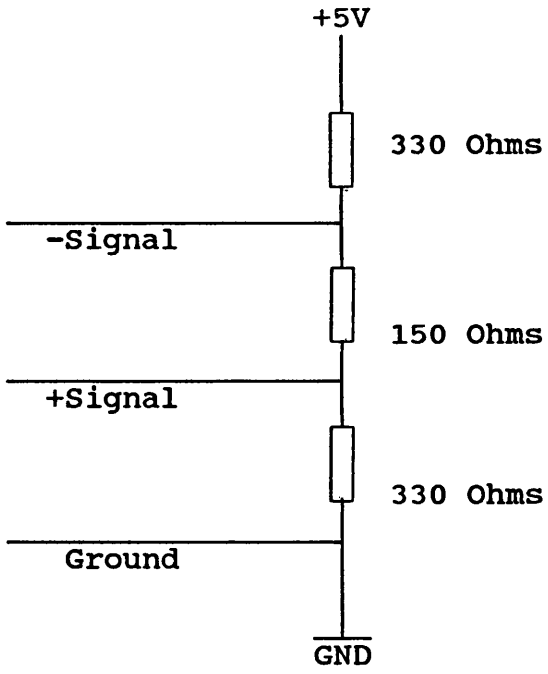
Differential				Single ended	
Pin number	Signal name	Pin number	Signal name	Pin number	Signal name
1	Shield GND.	2	GND	2	D0
3	+D0	4	-D0	4	D1
5	+D1	6	-D1	6	D2
7	+D2	8	-D2	8	D3
9	+D3	10	-D3	10	D4
11	+D4	12	-D4	12	D5
13	+D5	14	-D5	14	D6
15	+D6	16	-D6	16	D7
17	+D7	18	-D7	18	D(P)
19	+D(P)	20	-D(P)	20	GND
21	DIFFSENS	22	GND	22	GND
23	GND	24	GND	24	GND
25	TERMPWR	26	TERMPWR	26	TERMPWR
27	GND	28	GND	28	GND
29	+ATN	30	-ATN	30	GND
31	GND	32	GND	32	ATN
33	+BSY	34	-BSY	34	GND
35	+ACK	36	-ACK	36	BSY
37	+RST	38	-RST	38	ACK
39	+MSG	40	-MSG	40	RST
41	+SEL	42	-SEL	42	MSG
43	+C/D	44	-C/D	44	SEL
45	+REQ	46	-REQ	46	C/D
47	+I/O	48	-I/O	48	REQ
49	GND	50	GND	50	I/O

N.B. For single ended bus the odd pins are connected to ground.

Fig 5.3 Pin assignment of SCSI bus signals for differential and single ended modes.



Single-ended



Differential

Fig 5.4 Terminations on the SCSI bus signals.

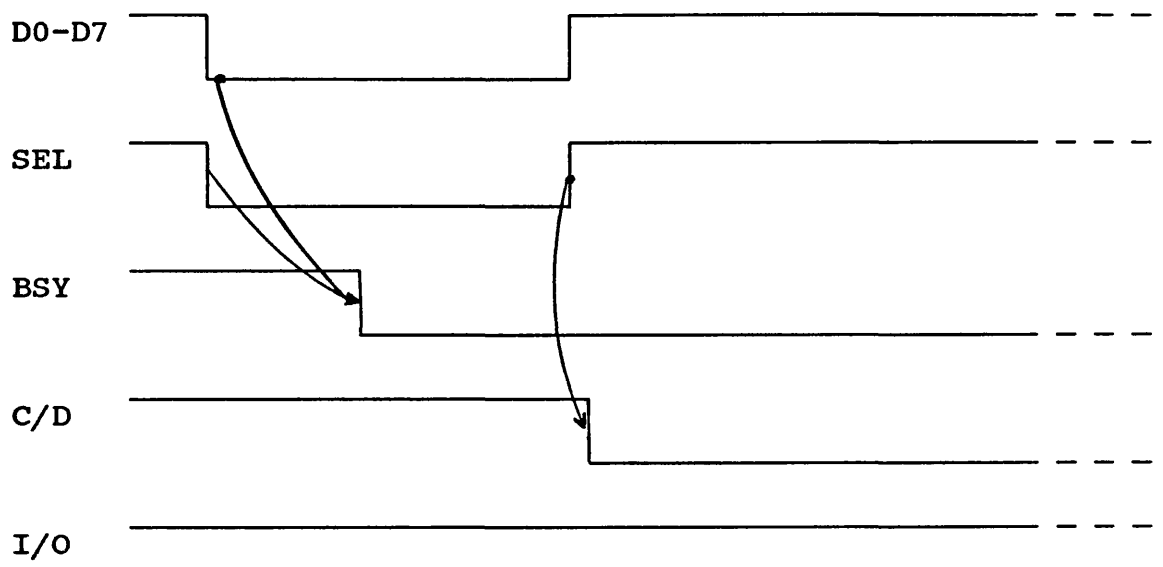


Fig 5.5 The device selection procedure in SCSI bus.

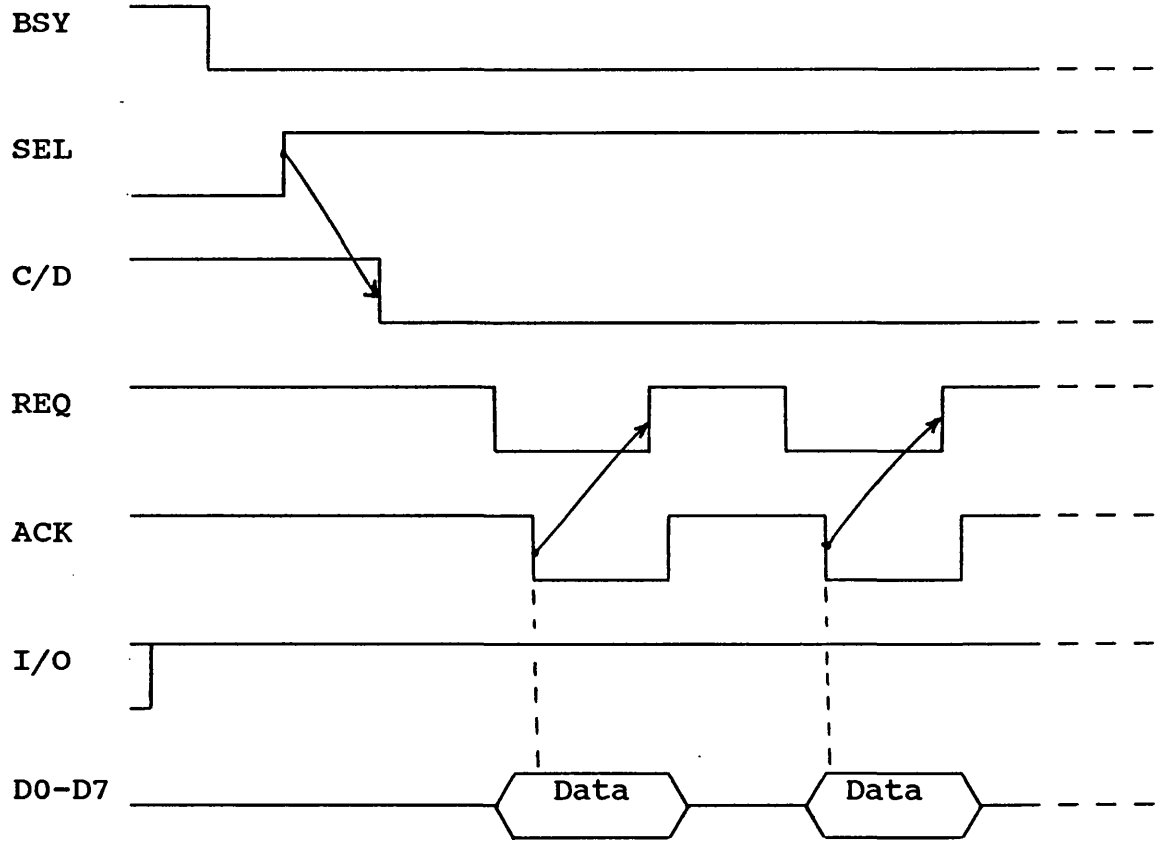
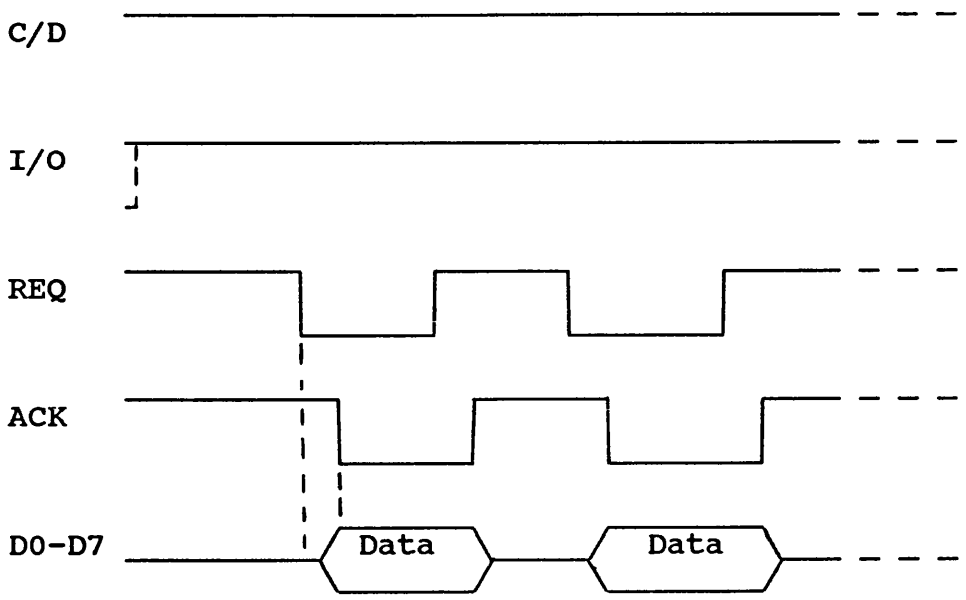
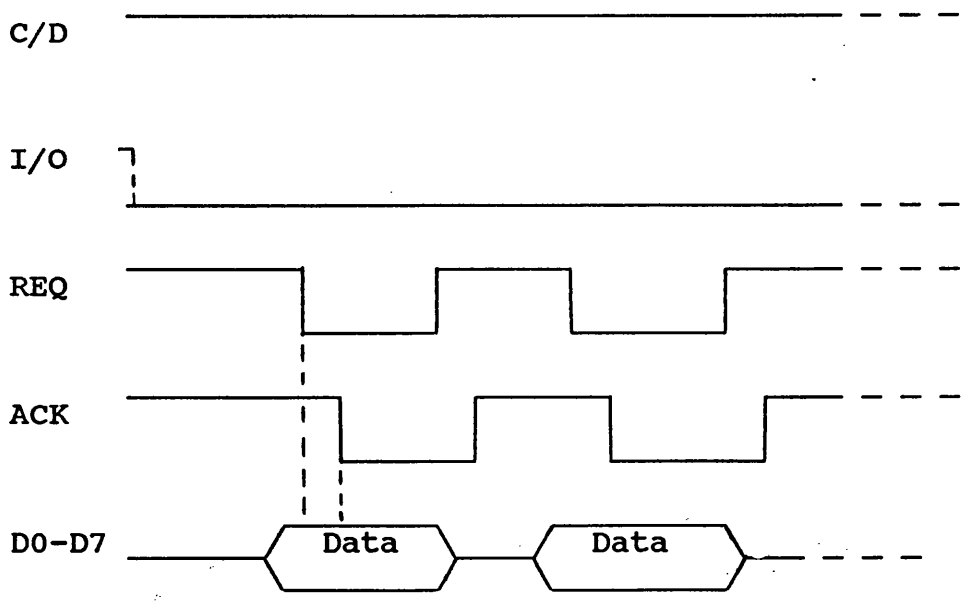


Fig 5.6 Transfer of command bytes with REQ/ACK handshake.



From initiator to target.



From target to initiator.

Fig 5.7 Data transfer on the SCSI bus.

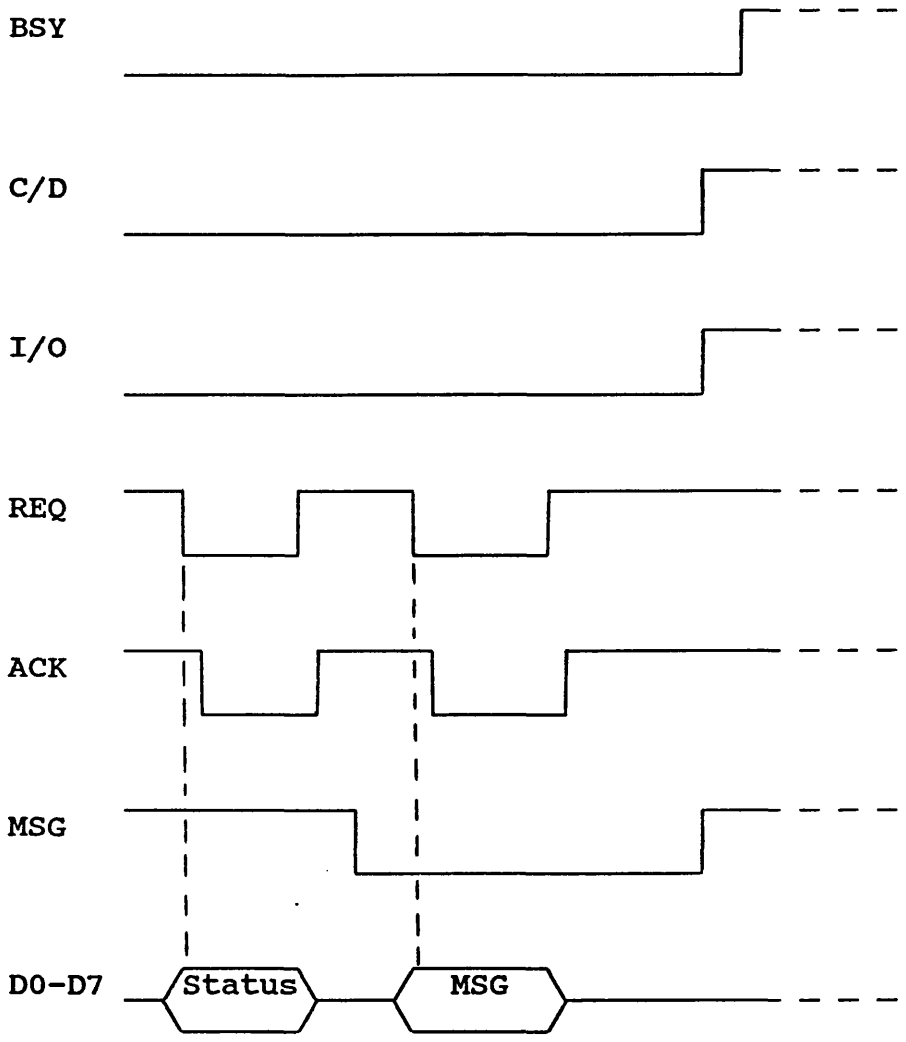


Fig 5.8 Status and message bytes transfer sequence on the SCSI bus.

Byte number	Description
0	Command code
1	Bit 0 to 4 logical block address (MSB) Bit 5 to 7 logical unit number
2	Logical block address
3	Logical block address (LSB)
4	Transfer length (blocks)
5	Control byte bit 0 for linked commands bit 1 flag for message

Byte number	Description
0	Command code
1	Bit 0 Relative address flag Bit 1 to 4 Reserved Bit 5 to 7 logical unit number
2	Logical block address (MSB)
3	Logical block address
4	Logical block address
5	Logical block address (LSB)
6	Reserved
7	Transfer length (MSB)
8	Transfer length (LSB)
9	Control byte, bit 0 for linked commands bit 1 flag for message

Byte number	Description
0	Command code
1	Bit 0 Relative address flag Bit 1 to 4 Reserved Bit 5 to 7 logical unit number
2	Logical block address (MSB)
3	Logical block address
4	Logical block address
5	Logical block address (LSB)
6	Reserved
7	Reserved
8	Reserved
9	Transfer length (MSB)
10	Transfer length (LSB)
11	Control byte, bit 0 for linked commands bit 1 flag for message

Fig 5.9 Command descriptor blocks for six, ten and twelve byte SCSI commands respectively.

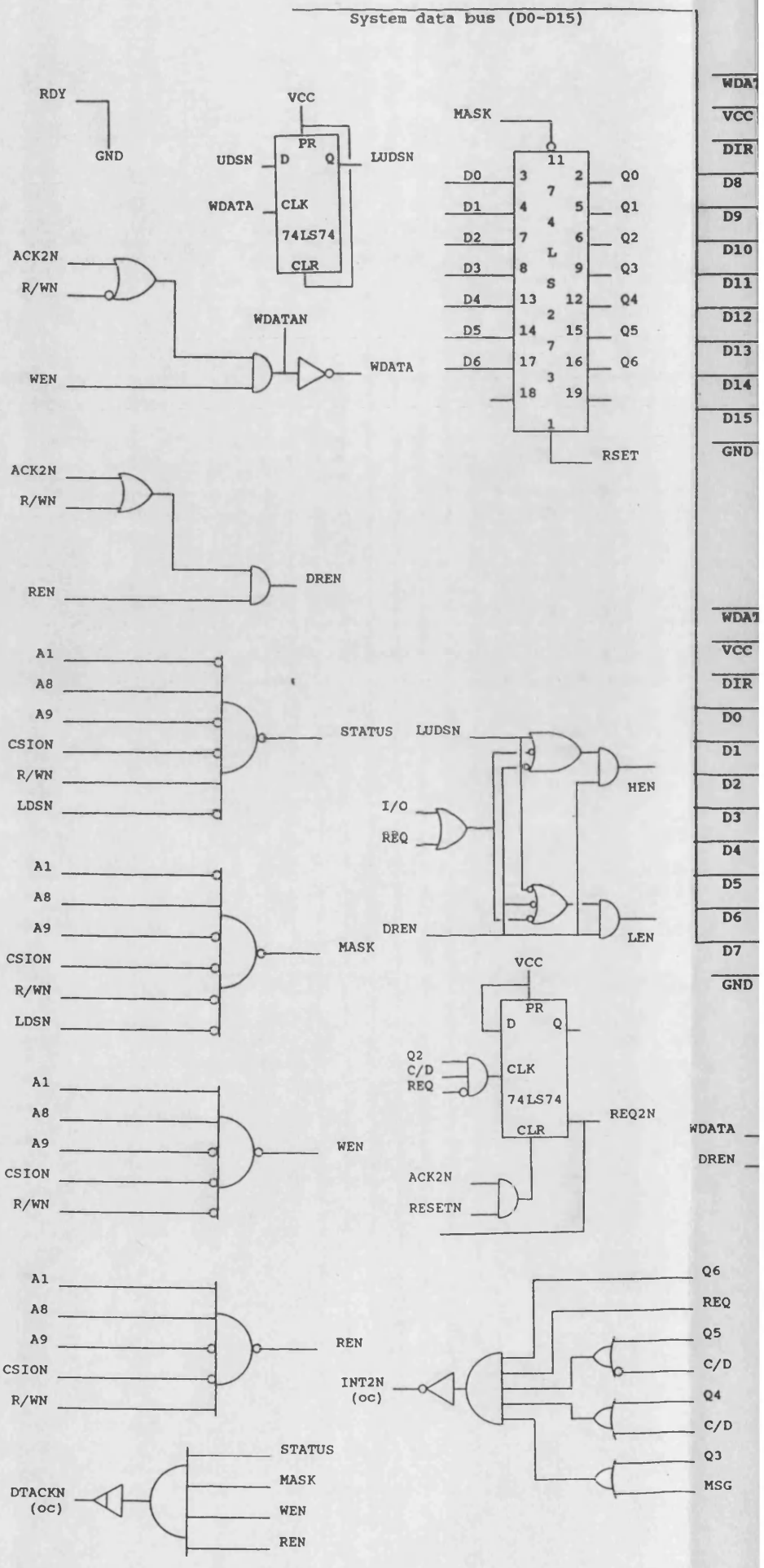


Fig 5.10 SASI interface circuit diagram.

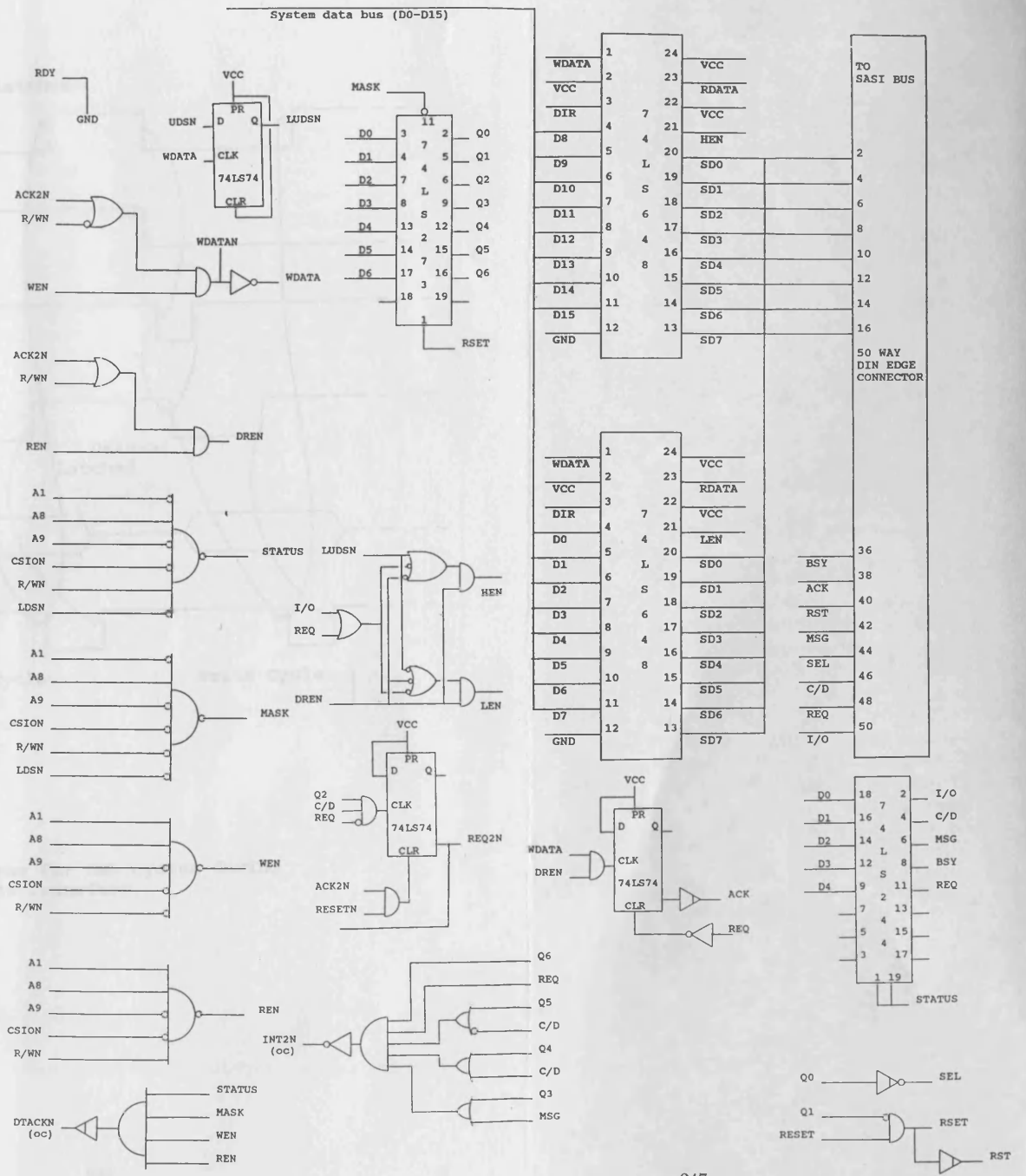


Fig 5.10 SASI interface circuit diagram.

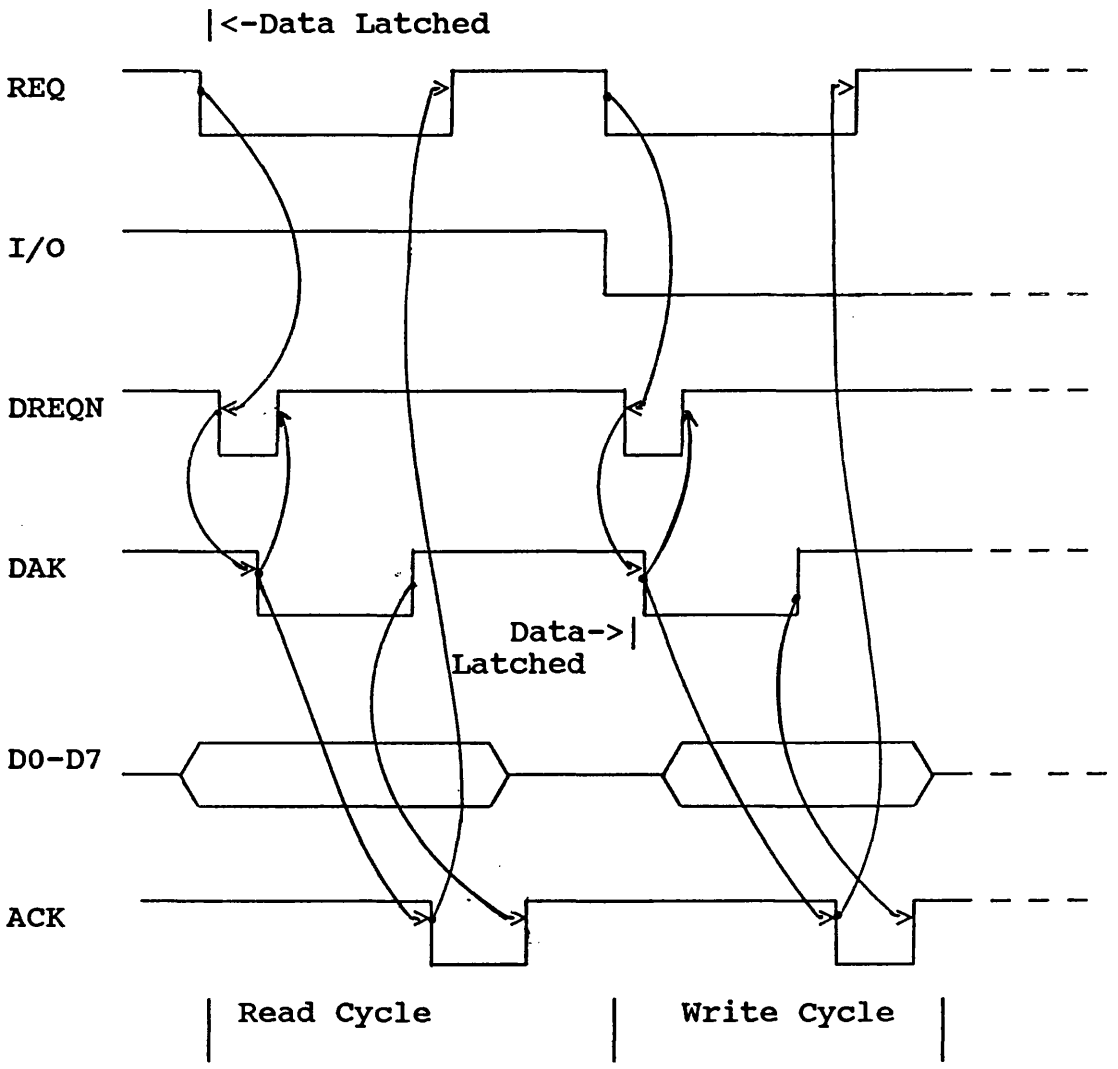


Fig 5.11 Timing diagram for DMA cycles during SASI bus data transfers.

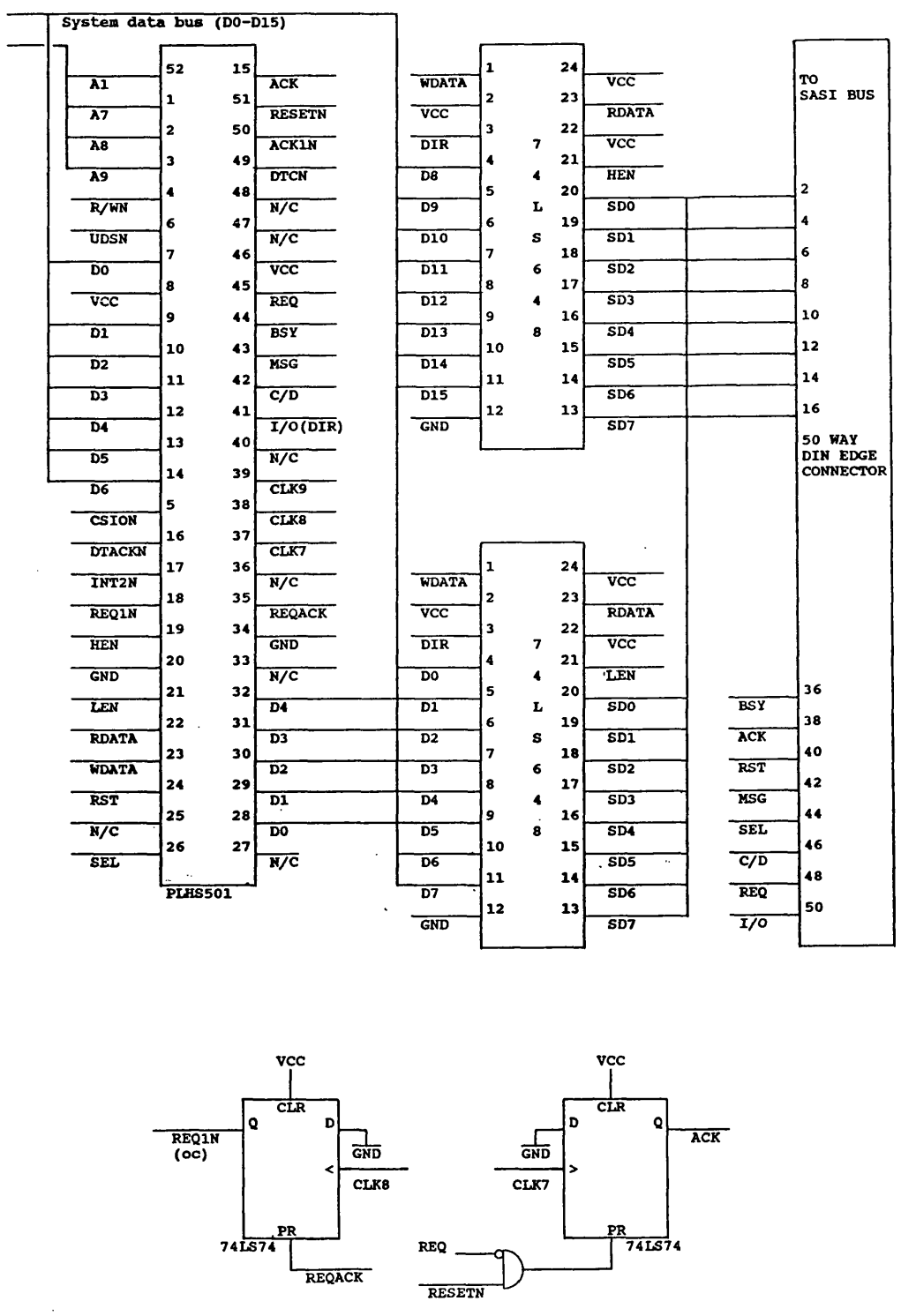


Fig 5.12 SASI interface circuit diagram using PLHS501.

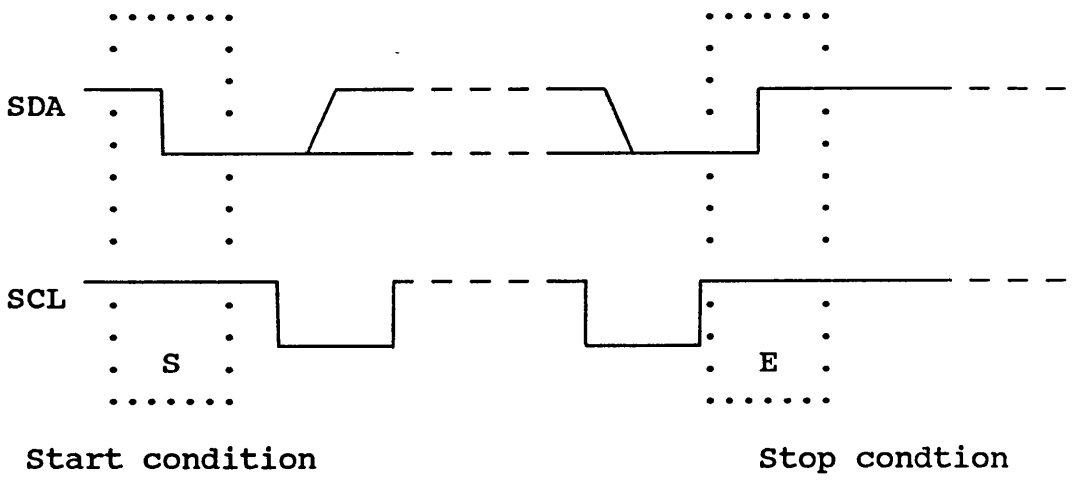


Fig 5.13 Start and Stop conditions on the I2C bus.

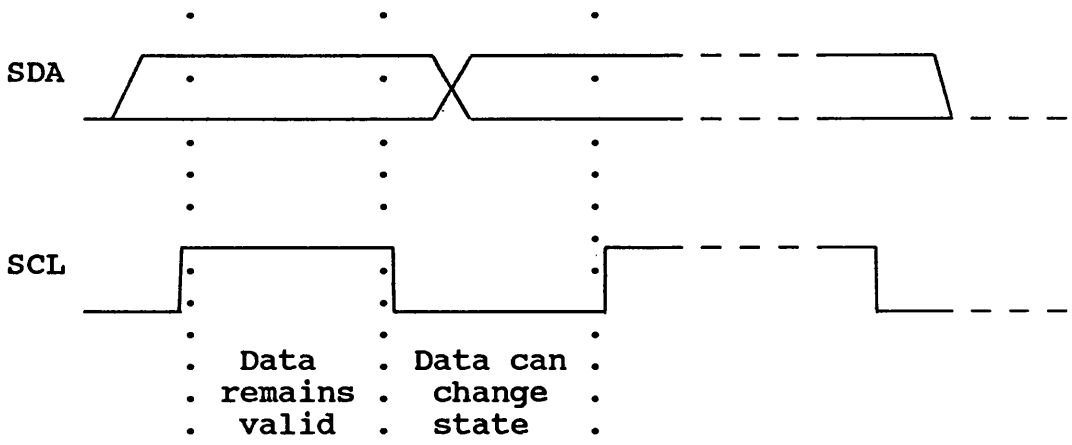


Fig 5.14 Bit transfer on the I2C bus.

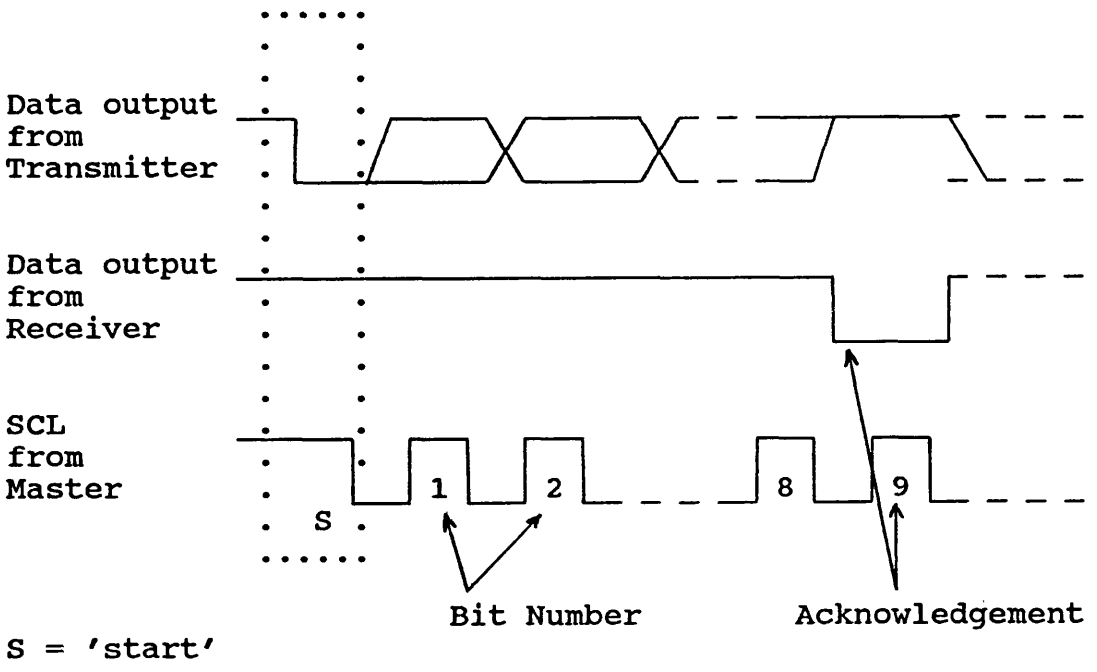


Fig 5.15 Acknowledge from receiver on the I2C bus.

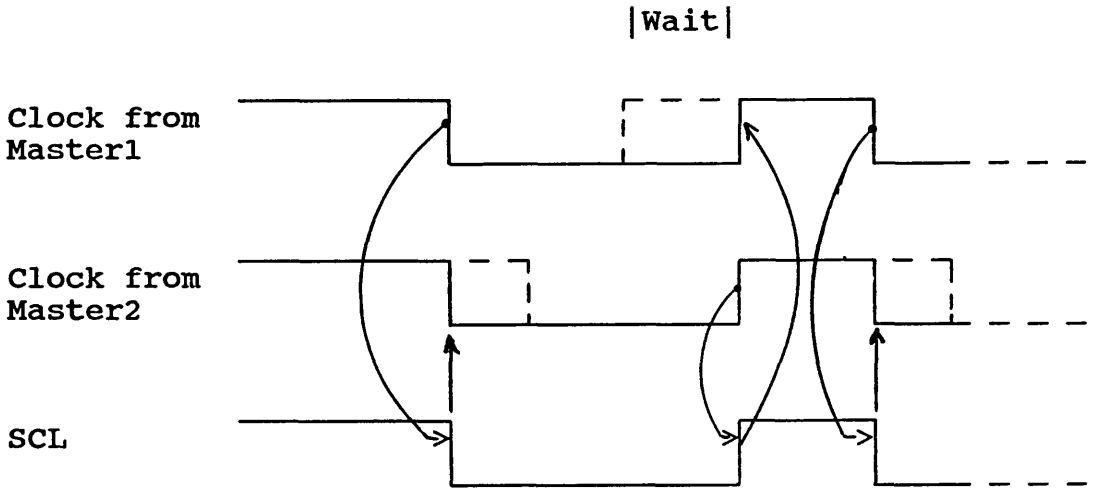


Fig 5.16 Clock synchronisation on I2C bus.

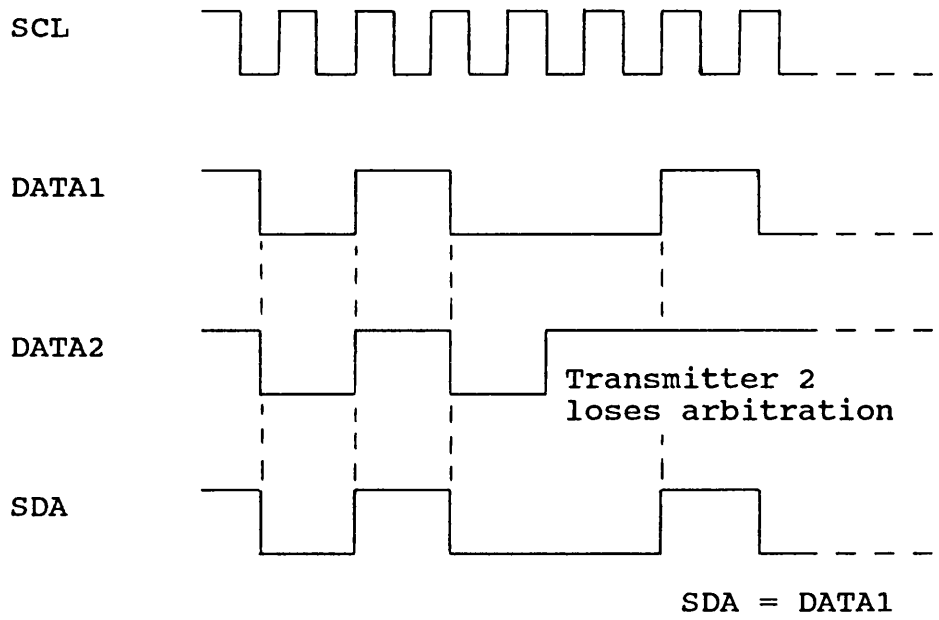


Fig 5.17 I2C bus arbitration procedure.

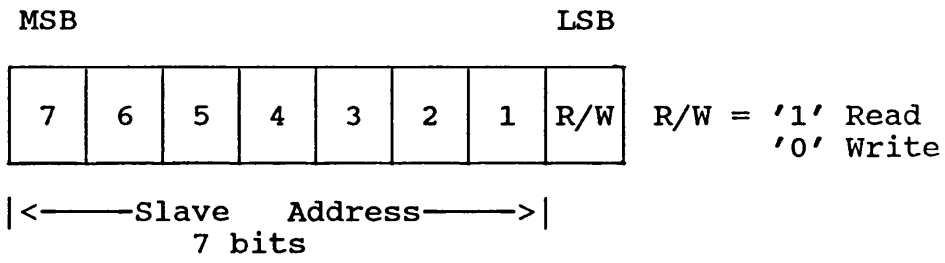
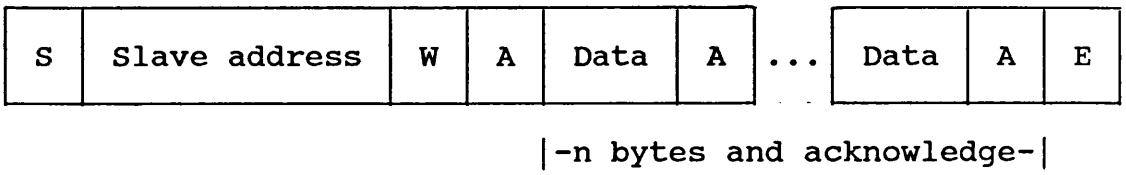
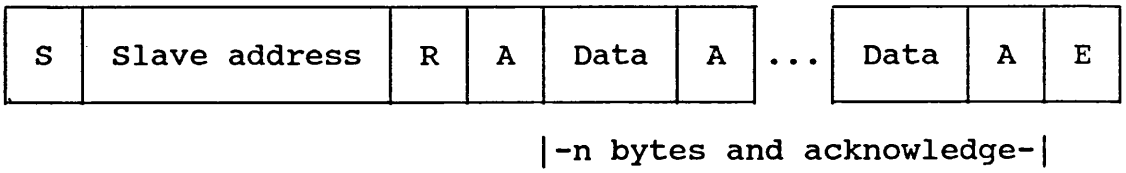


Fig 5.18 The first byte transmitted by the master after the start procedure.

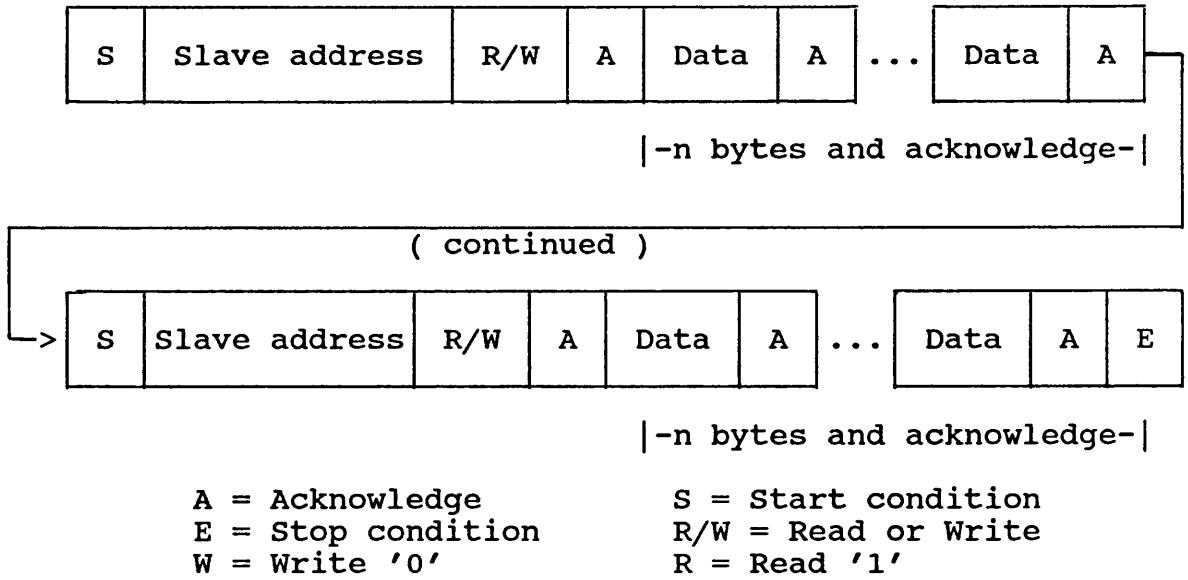
A: Master transmitting to slave receiver.



B: Master receiving from the slave transmitter.



C: Combined format.

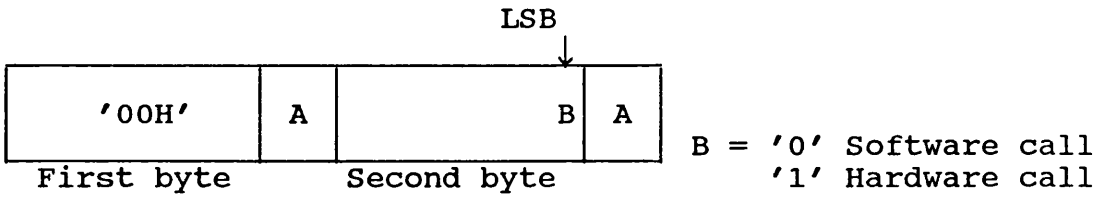


A = Acknowledge
E = Stop condition
W = Write '0'

S = Start condition
R/W = Read or Write
R = Read '1'

Fig 5.19 Possible data transfer formats on I2C bus.

General call address format:



Programming sequence from a programming master:

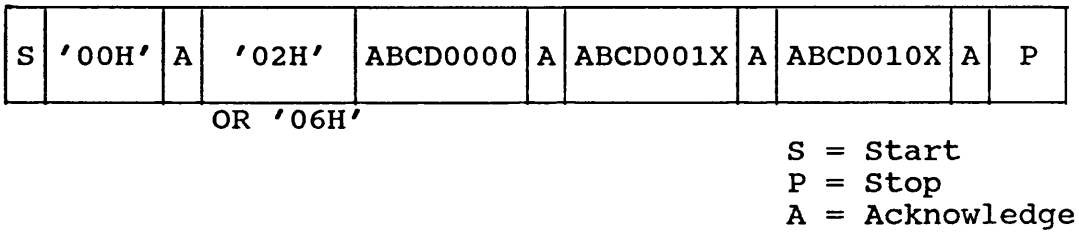


Fig 5.20 General call format and sequence of a programming master.

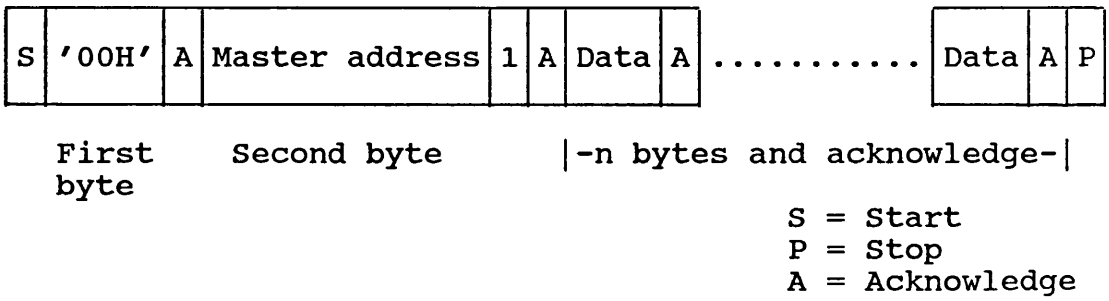


Fig 5.21 General call from a Hardware Master and data transfer from the Hardware Master transmitter.

I2C Address Register (IAR)

Bit	Function
B0	ALS, Always Selected
B1-B7	A0-A6, allocated slave address

I2C Status Register (ISR)

Bit	Function
B0	LRB, Acknowledge not received
B1	AD0, General call detected
B2	AAS, Addressed as slave
B3	AL, Arbitration lost
B4	PIN, No interrupt
B5	BB, Bus is busy
B6	TRX, Transmitter mode
B7	MST Master mode

I2C Control Register (ICR)

Bit	Function
B0	Not used
B1	Not used
B2	ACK, Acknowledge reception
B3	ESO, Enable I2C interface
B4	Not used
B5	Not used
B6	SEL, Selected
B7	Not used

Fig 5.22 Registers on the SCC68070
Inter-IC bus Interface.

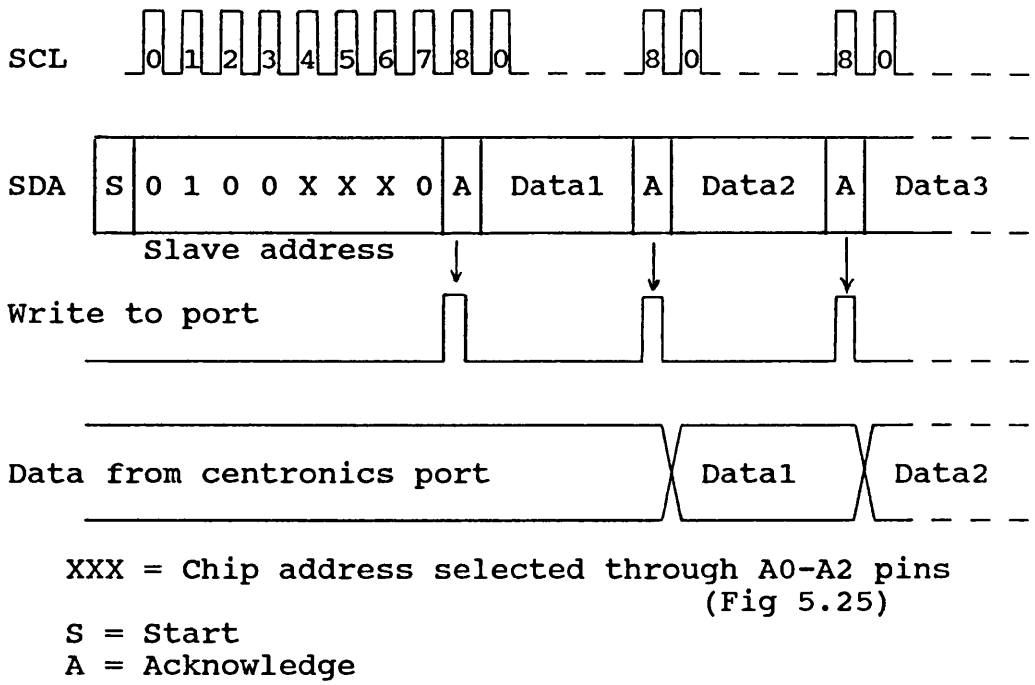


Fig 5.23 PCF8574 write mode (using as the output port).

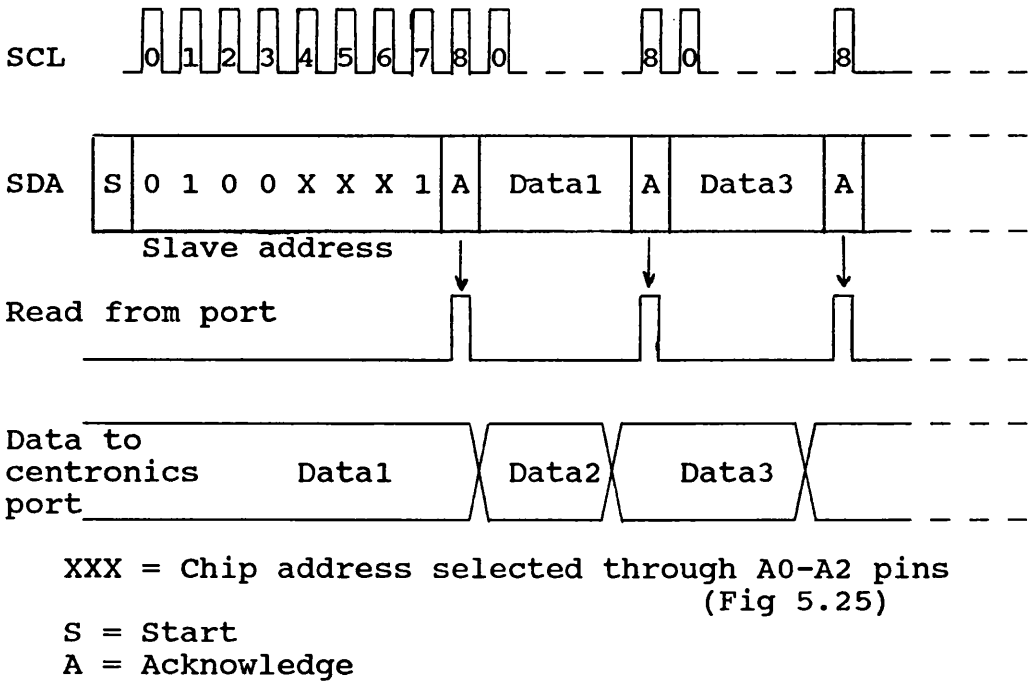


Fig 5.24 PCF8574 read mode (using as the input port).

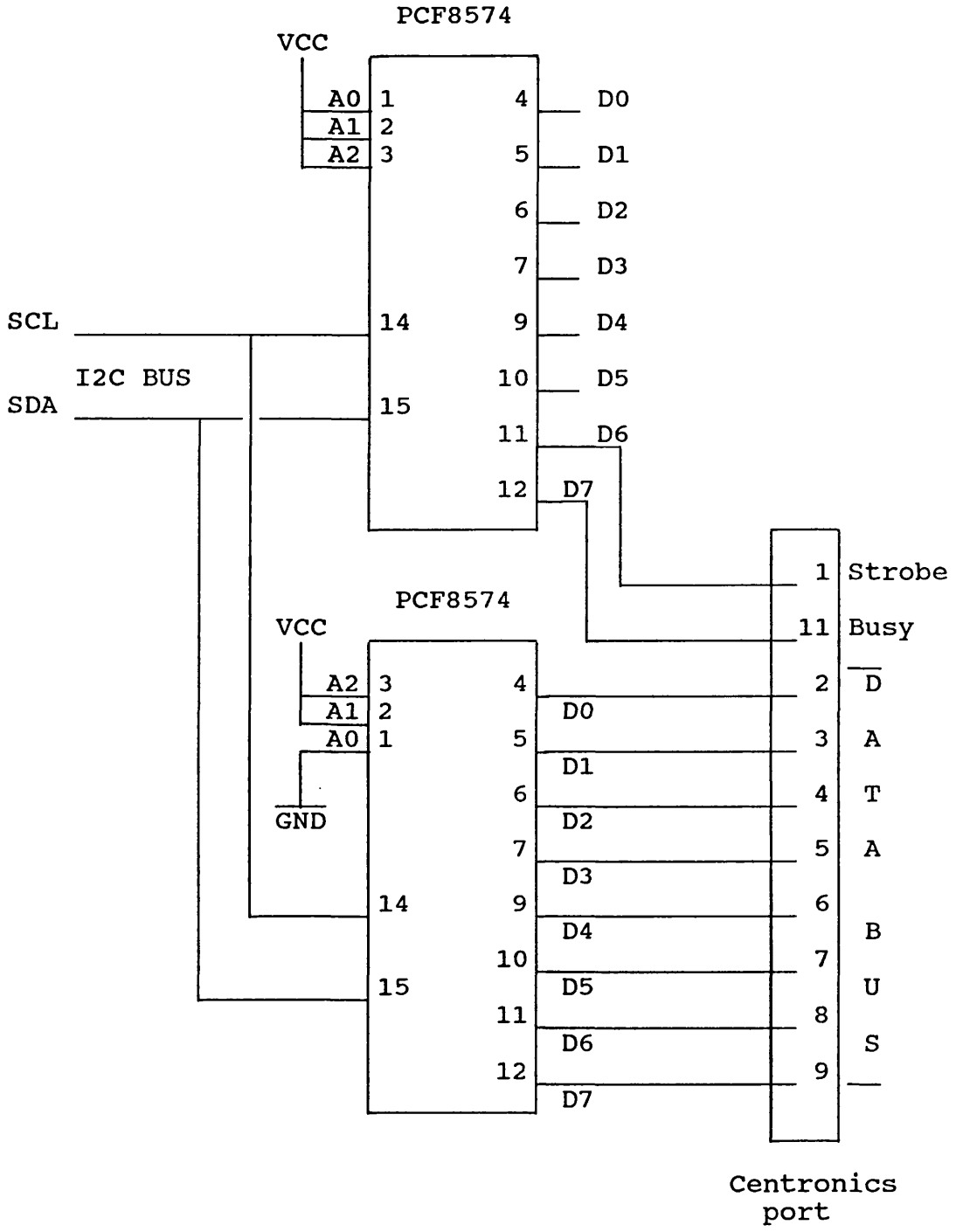


Fig 5.25 Printer interface circuit diagram.

RAM Address	Clock mode	Event counter mode
00	Control/Status	Control/Status
01	Hundredth of a Second (BCD)	Counter digit 0, 1 (BCD)
02	Seconds (BCD)	Counter digit 2, 3 (BCD)
03	Minutes (BCD)	Counter digit 4, 5 (BCD)
04	Hours (BCD)	Free
05	Year/Date	Free
06	Weekday/Month	Free
07	Timer (BCD)	Timer (BCD)
08	Alarm Control	Alarm Control
09	Alarm Hundredth of a second (BCD)	Alarm digit 0, 1 (BCD)
0A	Alarm Seconds (BCD)	Alarm digit 2, 3 (BCD)
0B	Alarm Minutes (BCD)	Alarm digit 4, 5 (BCD)
0C	Alarm Hours (BCD)	Free
0D	Alarm date	Free
0E	Alarm Month	Free
0F	Alarm Timer	Alarm Timer
10-FF	Free RAM	Free RAM

Fig 5.26 Register arrangement in PCF8583.

Address = 00

Bit	Function
B0	Timer flag, if alarm disabled then 50% duty factor seconds flag
B1	Alarm flag, if alarm disabled then 50% duty factor minutes flag
B2	Alarm Enable bit
B3	Mask flag; if set date and month are read directly
B4-B5	Function mode 0 = clock mode (32.768 KHz) 1 = clock mode (50 Hz) 2 = Event counter mode 3 = Test mode, Reserved
B6	Enable capture, Hold last count
B7	Stop counting flag, reset divider

Fig 5.27 Control and status register in PCF8583.

Hours register,
Address = 04, reset to 00H

Bit	Function
B0-B3	Unit hours (BCD)
B4-B5	ten hours (0 to 2)
B6	AM/PM flag (0 = AM, 1 = PM)
B7	Time format (0 = 24 hour, 1 = 12 hours)

Year/date register,
Address = 05, reset to 01H

Bit	Function
B0-B3	Unit days (BCD)
B4-B5	Ten days (BCD)
B6-B7	Year number (0 to 3, read as 0 if mask enabled)

Weekdays/month register,
Address = 06, reset to 01H

Bit	Function
B0-B3	Unit months (BCD)
B4	Ten months
B5-B7	Weekday number (0 to 6, read as 0 if mask enabled)

Timer register,
Address = 07, reset to 00H

Bit	Function
B0-B3	Unit days (BCD)
B4-B7	Ten days (BCD)

Fig 5.28 Clock/timer registers in PCF8583.

Clock mode:
Address = 08, reset to 00H

Bit	Function
B0-B2	Timer functions 0 = no timer 1 = hundredth of a second 2 = second 3 = minute 4 = hour 5 = days 6 = not used 7 = test mode
B3	Timer interrupt enable
B4-B5	Clock alarm functions 0 = no clock alarm 1 = daily alarm 2 = weekday alarm 3 = dated alarm
B6	Timer alarm enable
B7	Alarm interrupt enable

Event counter mode:
Address = 08, reset to 00H

Bit	Function
B0-B2	Timer functions 0 = no timer 1 = units 2 = 100 3 = 10 000 4 = 1 000 000 5 = not used 6 = not used 7 = test mode
B3	Timer interrupt enable
B4-B5	Event alarm functions 0 = no event alarm 1 = event alarm 2 = not used 3 = not used
B6	Timer alarm enable
B7	Alarm interrupt enable

Fig 5.29 Alarm control register in clock and event counter modes.

Alarm weekday register,
Address = 0EH, reset to 00H

Bit	Function
B0	Enable alarm for weekday 0
B1	Enable alarm for weekday 1
B2	Enable alarm for weekday 2
B3	Enable alarm for weekday 3
B4	Enable alarm for weekday 4
B5	Enable alarm for weekday 5
B6	Enable alarm for weekday 6
B7	Not used

Fig 5.30 Selection of weekdays for alarm.

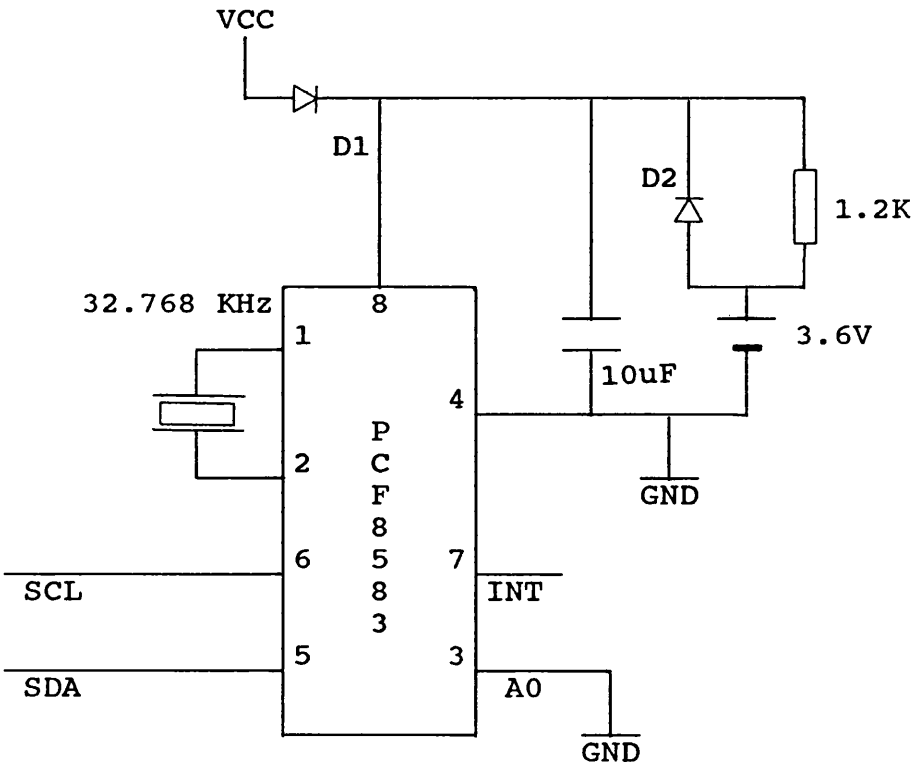
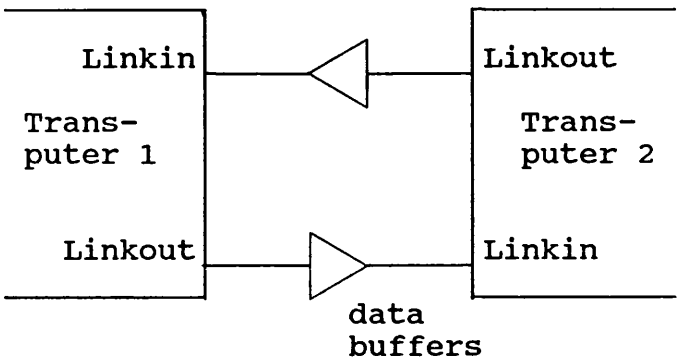
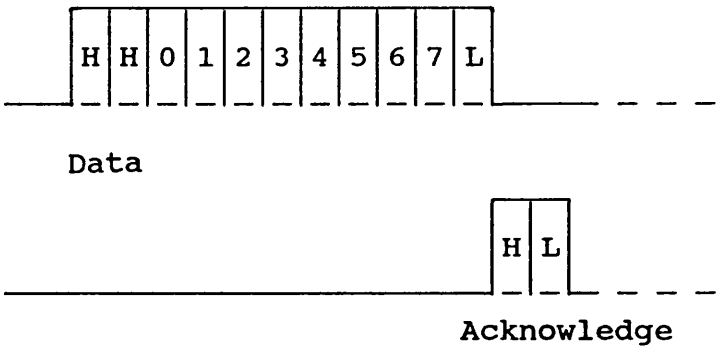


Fig 5.31 Circuit diagram for PCF8583.



a: Link connection between transputers.



b: Data transmission and acknowledge
on the transputer link.

Fig 5.32 Protocol in transputers via links.

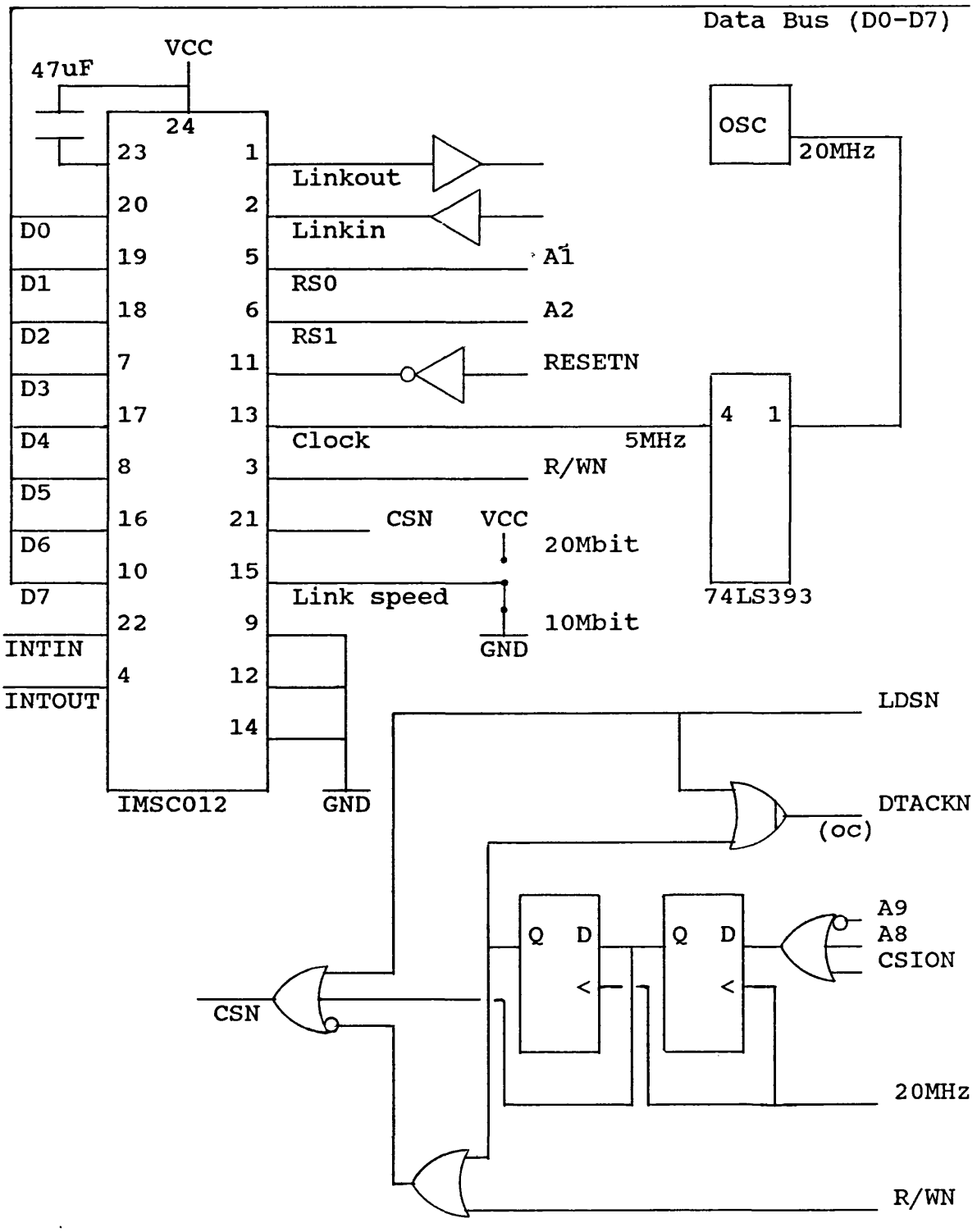


Fig 5.33 Link Adapter circuit diagram.

```
LONG  : Packet link.

LONG  : Device ID.

LONG  : Packet type.

LONG  : Packet res1.

LONG  : Packet res2.

LONG  : Drive unit number.

LONG  : Buffer address.

LONG  : Operation size.

LONG  : Starting offset.
```

Fig 6.1 A disk device packet under Tripos.

```
Open( DCB )
{    DeviceID = DCB[ID];
     issue four 'interrupt sense' commands;
     issue 'mode' command;
     issue 'specify' command;
     for( drive = 0) to all available disk drives )
      {    issue reset command for drive;
           issue interrupt sense command
           and clear interrupt;
      }
     reset DMA( channel 2 );
     switch on interrupts in LIR;
}
```

Fig 6.2 The Open routine in the floppy disk driver.

```
reset DMA( channel 2 )
{    force abort DMA action;
     wait until status register 'cleared';
     initialise DMA;
     wait until ready;
     clear DMA device address register;
}
```

Fig 6.3 The subroutine ti initialise the DMA channel 2.

```
Close( DCB )
{    switch off interrupts in LIR;
}
```

Fig 6.4 The Close routine in the floppy disk driver.

```
StartIO( DCB, packet )
{    for ever
     {     packet action = packet[type];
           switch( packet action )
           {    case 'read'   : read( packet );
                                return;
                case 'write'  : write( packet );
                                return;
                case 'reset'  : reset( packet );
                                return;
                case 'format' : format( packet );
                                return;
                case 'status' : status( packet );
                                break;
                case 'motor'  : motor( packet );
                                break;
                default       :   packet[res1]  =   unknown error;
                                return packet( packet );
           }
      packet = next packet;
      if( next packet == 0 ) break;
     }
 }
```

```
Start Command( Command buffer )
{     wait until 'TAS flag' cleared;
      set 'TAS flag';
      set DMA controller to make data transfers;
      issue command to FDC;
}
```

Fig 6.5 The StartIO routine in the floppy disk driver.

```
read( packet )
{     get drive unit;
      get buffer address;
      get read offset;
      get read size;
      calculate track number,
      sector number and head number
      for the floppy disk drive;
      make command buffer for read command;
      Start Command( command buffer );
}
```

Fig 6.6 The subroutine to read from to a floppy disk.

```
write( packet )
{     get drive unit;
      get buffer address;
      get write offset;
      get write size;
      calculate track number,
      sector number and head number
      for the floppy disk drive;
      make command buffer for write command;
      Start Command( command buffer );
}
```

Fig 6.7 The subroutine to write to a floppy disk.

```
format( packet )
{     get drive unit;
      get format offset;
      calculate track number and
      head number for the floppy disk drive;
      make data buffer to put on the disk during format;
      make command buffer for format command;
      make seek command buffer for seek command;
      set flag for seek command;
      issue seek command to FDC;
}
```

Fig 6.8 The subroutine to format a floppy disk.

```
ResetAll( drive )
{     issue 'reset drive' command to FDC;
      issue interrupt sense command to FDC
      to clear interrupt from reset command;
      if( 'error detected' ) Error = 'reset error';
                        ELSE Error = 0;
}
```

Fig 6.9 The ResetAll routine in the floppy disk driver.

```
Status( packet )
{     set unit ready in the packet;
      return packet( packet );
}
```

Fig 6.10 The Status routine in the floppy disk driver.

```
Motor( packet )
{     return packet( packet );
}
```

Fig 6.11 The Motor routine in the floppy disk driver.

```
return packet( packet )
{     unlink the packet from WorkQ;
      adjust the WorkQ tail;
      store packet address in SCC68070 register 'D1';
      store DeviceID in SCC68070 register 'D2';
      make the 'Kernel call' to return packet;
}
```

Fig 6.12 The subroutine to return a packet to the Tripos.

```
AbortIO( DCB, packet )
{    no operation;
}
```

Fig 6.13 The AbortIO routine in the floppy disk driver.

```
Interrupt()
{     if( controller is in 'not ready to read' state )
      {     /* a reset or seek command */
            issue interrupt sense command
            and read command status;
            if( error detected )
            {     Error = type of error;
                  return(1);
            }
            if( seek flag )
            {     clear seek flag;
                  Start Command( command buffer );
                  return(0);
            }
            else return(1);
      }
      /* read write and format commands */
      read command result bytes from the FDC;
      if( 'error detected' )
      {     Error = error type;
            clear TAS flag;
            if( try flag ) return(1);
```

```
            set try flag;
            reset drive to track 0;
            reset DMA( channel 2 );
            Start Command( command buffer );
            return(0);
        }
        else
        {   Error = 0;
            clear TAS flag;
            return(1);
        }
        return(-1);
    }
```

Fig 6.14 The interrupt routine in the floppy disk driver.

```
RecallIO( DCB, itb )
{    extract packet address from DCB;
     if( Error )
          {    packet[res1] = Error;
               reset DMA( channel 2 );
          }
          else
          {    packet[res2] = 0;
               packet[res1] = data transfer length;
          }
     return packet( packet );
     clear try flag;
}
```

Fig 6.15 The RecallIO routine in the floppy disk driver.

```
install tripos()
{     read 'name of the disk device',
      'file name' to be installed and
      'version number' from console;
      ID = set device( name of the disk device );
      lock = locateobject( file name );
      get 'key' for the file;
      buffer = get block zero from the disk device;
      buffer[version] = key;
      write modified buffer on block zero of the disk;
}

set device( device name )
{     ID = find device if already mounted;
      if( ID == 0 )
      {     load device code;
            create device and link it in Devtab;
            ID = ID of the created device;
      }
      return( ID )
}
```

Fig 6.16 Installing of Tripos image file.

```
Bootstrap()
{     initialise SCC68070 and VSC;
      initialise stack and store;
      buffer = read buffer address;
      block number = 0;
      get device type from console;
      get version from console;
      initialise device( device );
      read block( block number );
      block number = buffer[version];
      {     read block( block number );
            if( checksum( buffer ) == 0 ) break;
      } repeat;
      block number = buffer[next block];
      {     read block( block number );
            while( checksum( buffer ) != 0 )
                  read block( block number );
            copy block data into store and
            increment store pointer;
            block number = buffer[next block];
      } repeat until end of file;
      point to the beginning of store pointer
      and run bootstrap;
 }
```

```
read block( block number)
{     calculate track number,
      sector number and head number
      for the floppy disk drive;
      make command buffer for read command;
      {     issue read command to FDC;
            read data from FDC in data phase;
            read result bytes in result phase;
            if( no error detected ) break;
      } repeat;
}

checksum( buffer )
{     for( i = 0 to buffer length )
      sum = buffer[i];
      return(sum);
}
```

Fig 6.17 The flow chart for the bootstrap.

```
LONG  : Packet link.

LONG  : Device ID.

LONG  : Packet type.

LONG  : Packet res1.

LONG  : Packet res2.

LONG  : Drive unit number.

LONG  : Buffer address.

LONG  : Operation size.

LONG  : Slave device address.
```

Fig 6.18 I2C device packet type:

```
set-time()
{     get days, mins and ticks from rootnode;
      device ID = set device( 'I2C' );
      stop 'clock function' of the chip;
      store time in on-chip RAM;
      start 'clock function' of the clock chip;
}
```

Fig 6.19 Setting time into the PCF8583.

```
restore-time()
{     device ID = set device( 'I2C' );
      read elapsed time and stored time from clock chip;
      make present days, mins and ticks;
      update time in the rootnode;
}
```

Fig 6.20 Restoring time from the PCF8583.

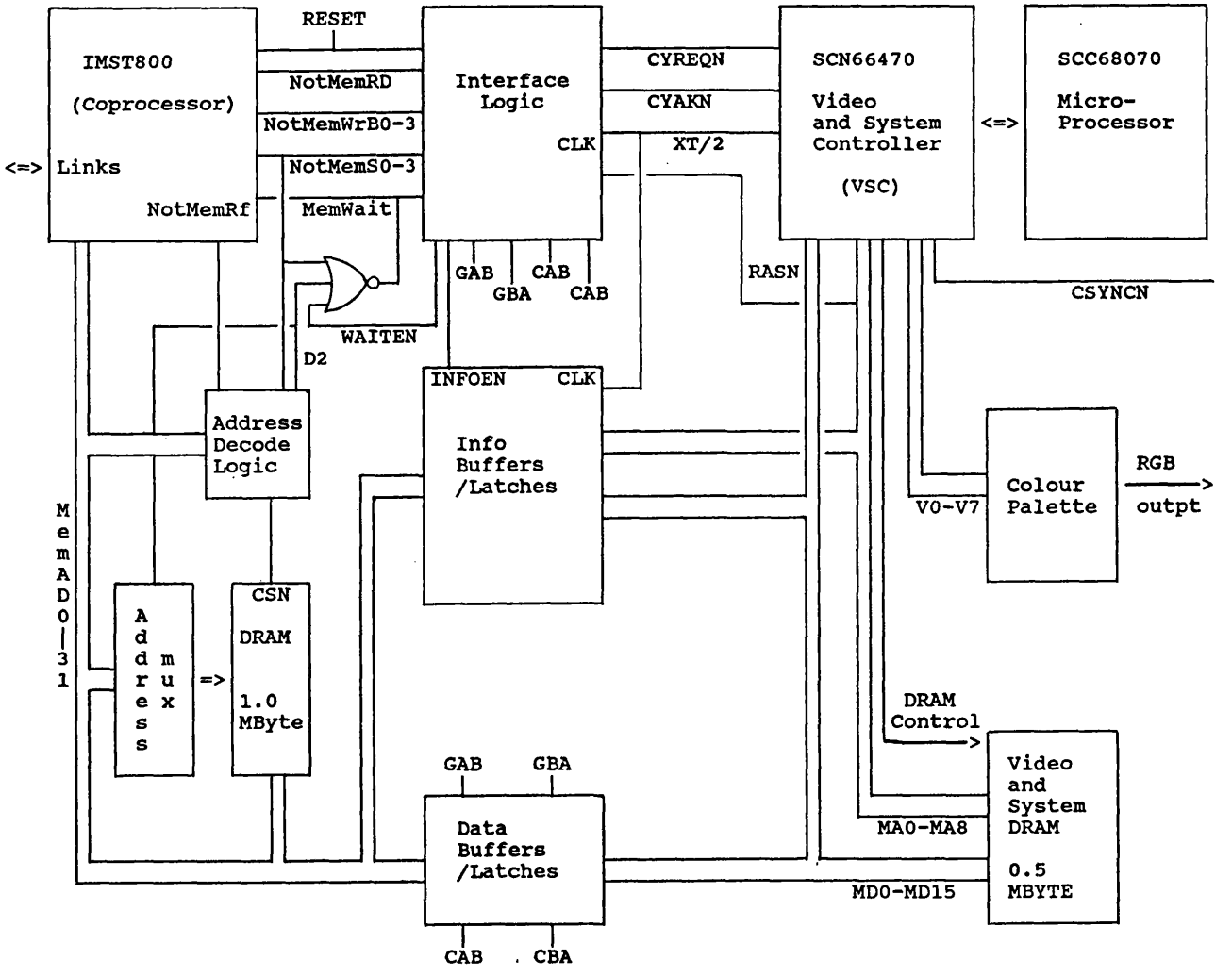


Fig 7.1 Block diagram of the Graphics Processing board.

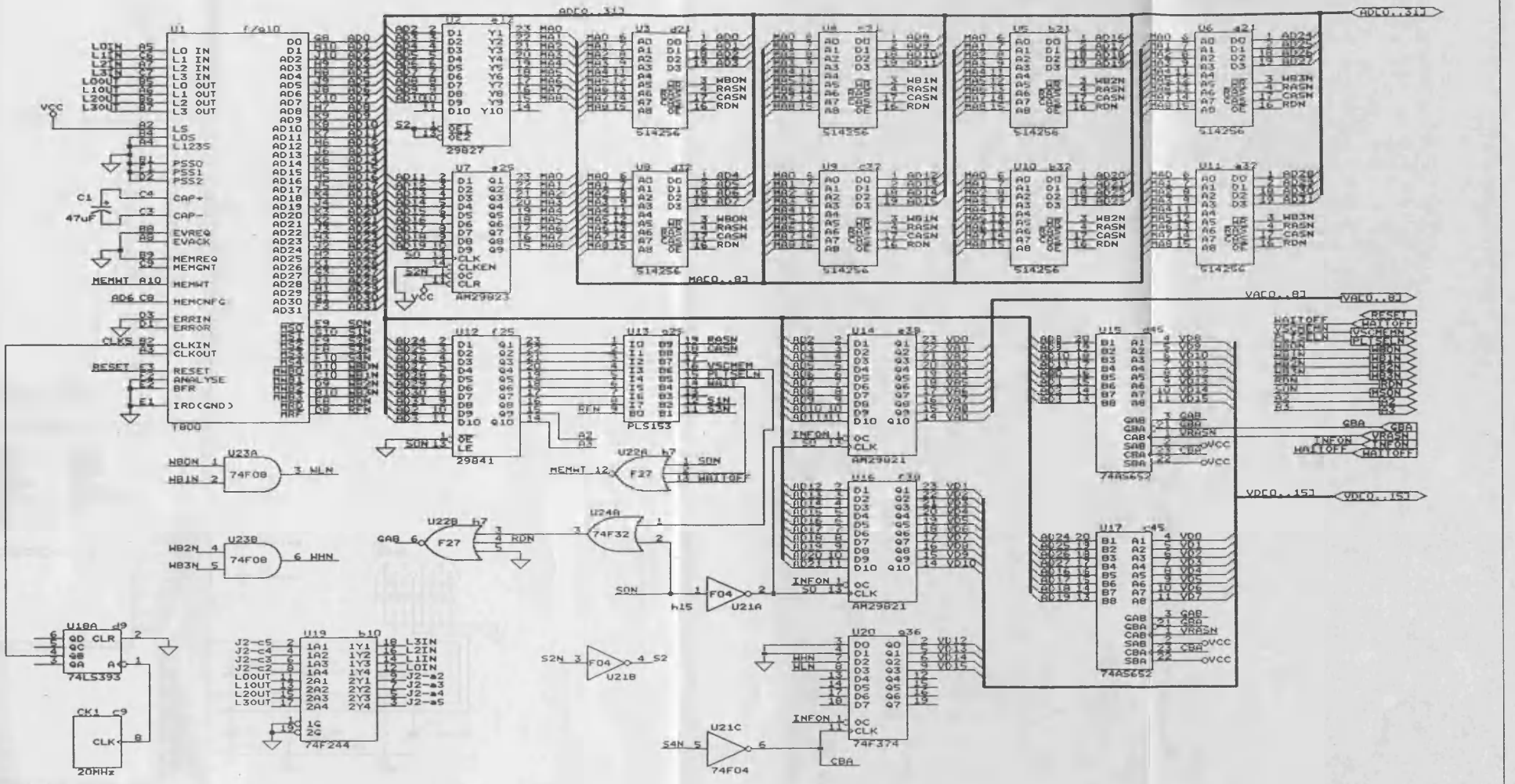


Fig 7.2 Graphics processing board circuit diagram.

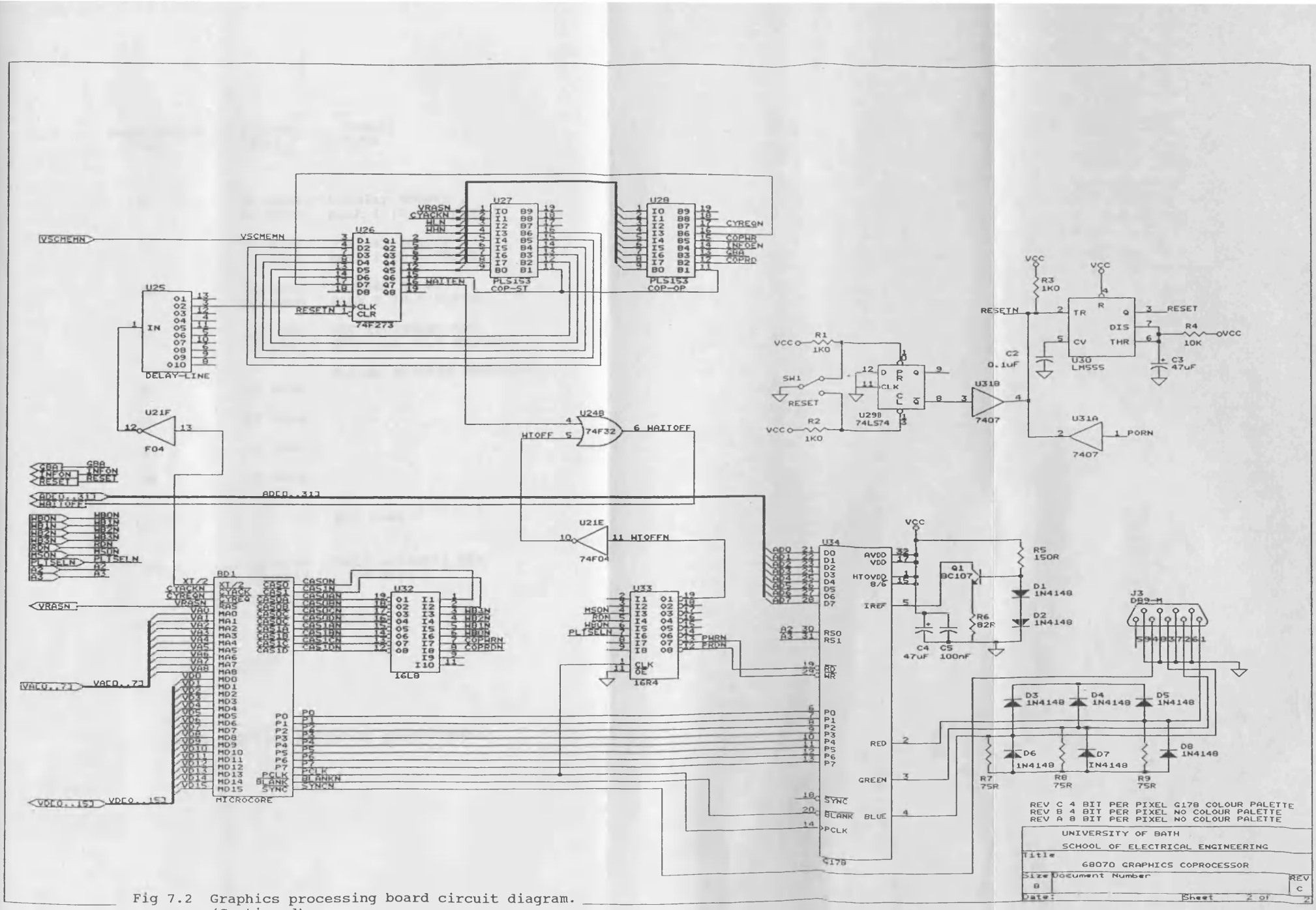


Fig 7.2 Graphics processing board circuit diagram. (Continued)

Page Number (Hex)	Address (Hex)	Memory Type
00	00 0000 07 FFFF	Display Memory Bank 1 (0.5 Mbyte)
00	08 0000 0F FFFF	Display Memory Bank 2 (0.5 Mbyte)
00	10 0000 17 FFFF	Display Memory Bank 3 (0.5 Mbyte)
00	18 0000 1F FFFF	VSC registers and I/O area (0.5 Mbyte)
		Colour palette registers
01	00 0000	
01	00 0004	
01	00 0008	
01	00 000B	
02-7F		Not Used
80	00 0000 0F FFFF	T800 external RAM (1.0 Mbyte)
81-FF		Not Used

Fig 7.3 T800 graphics processor memory map.

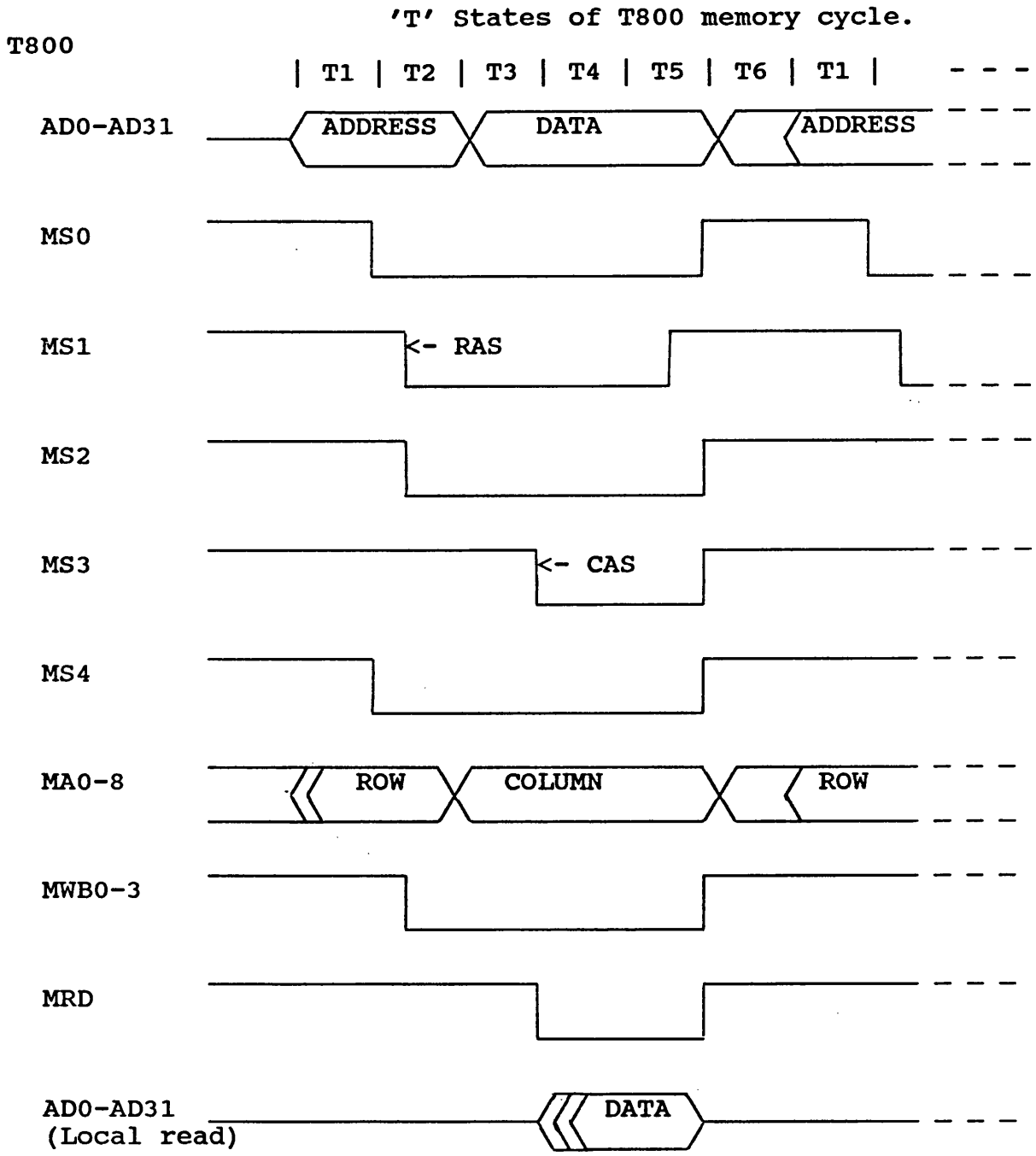
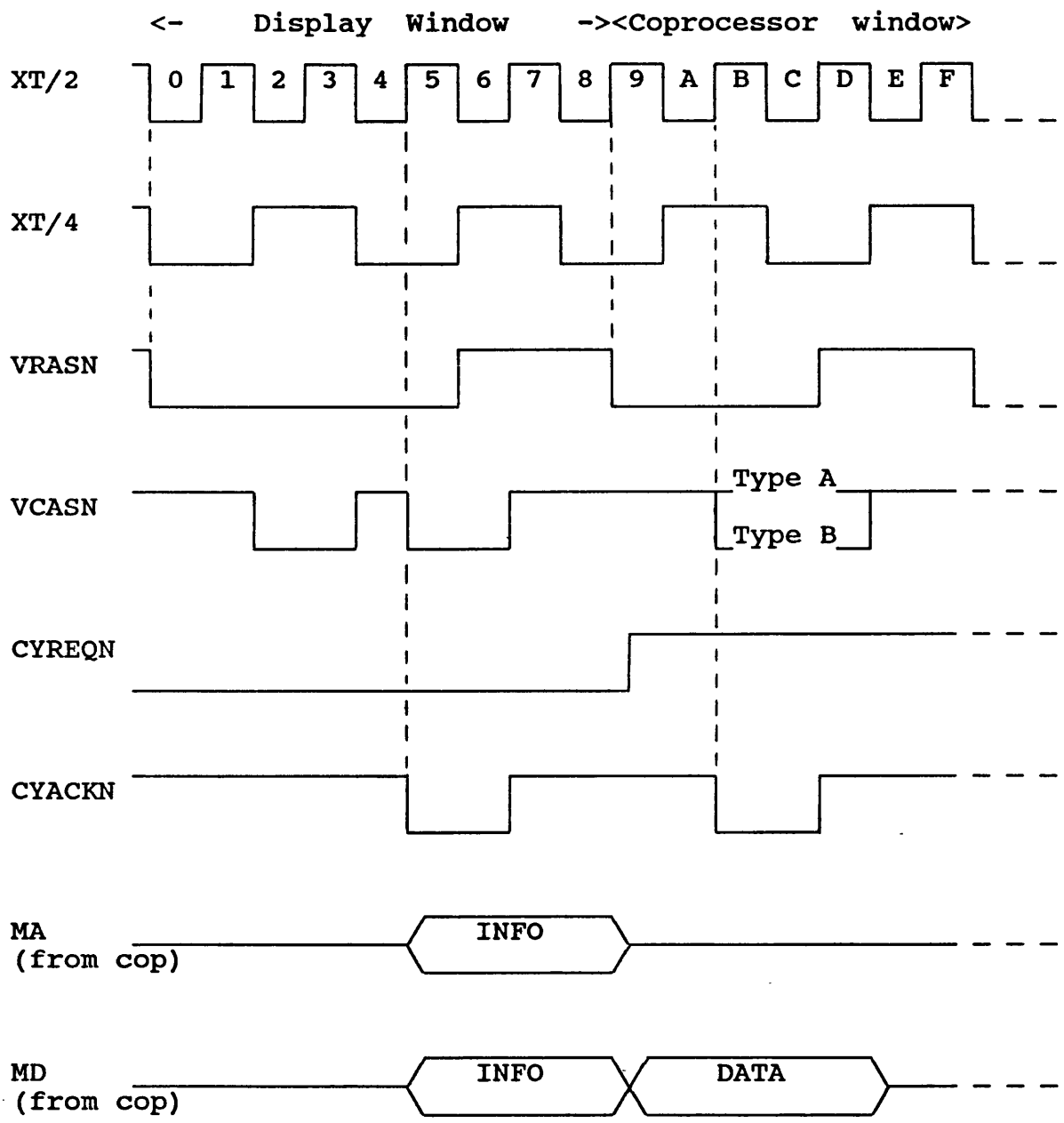


Fig 7.4 Timing diagram for T800 memory interface in 'MemAD6' memory configuration.

From Coprocessor (info)	256K DRAM Memory Bus (Normal Mode)
Address	
A1	MA0
A2	MA1
A3	MA2
A4	MA3
A5	MA4
A6	MA5
A7	MA6
A8	MA7
A9	MA8
A10	MD0
A11	MD1
A12	MD2
A13	MD3
A14	MD4
A15	MD5
A16	MD6
A17	MD7
A18	MD8
A19	MD9
A20	MD10
Not Used	MD11
SELA (Pixel accelerator reg A)	MD12
SELB (Pixel accelerator reg B)	MD13
R/WN (Lower byte)	MD14
R/WN (higher byte)	MD15

Fig 7.5 Information required by the VSC for the coprocessor cycle.



Type A = data transfer between VSC registers and coprocessor.
Type B = data transfer between display memory and coprocessor.

Fig 7.6 Distribution of the VSC memory cycle into the display window and the coprocessor window.

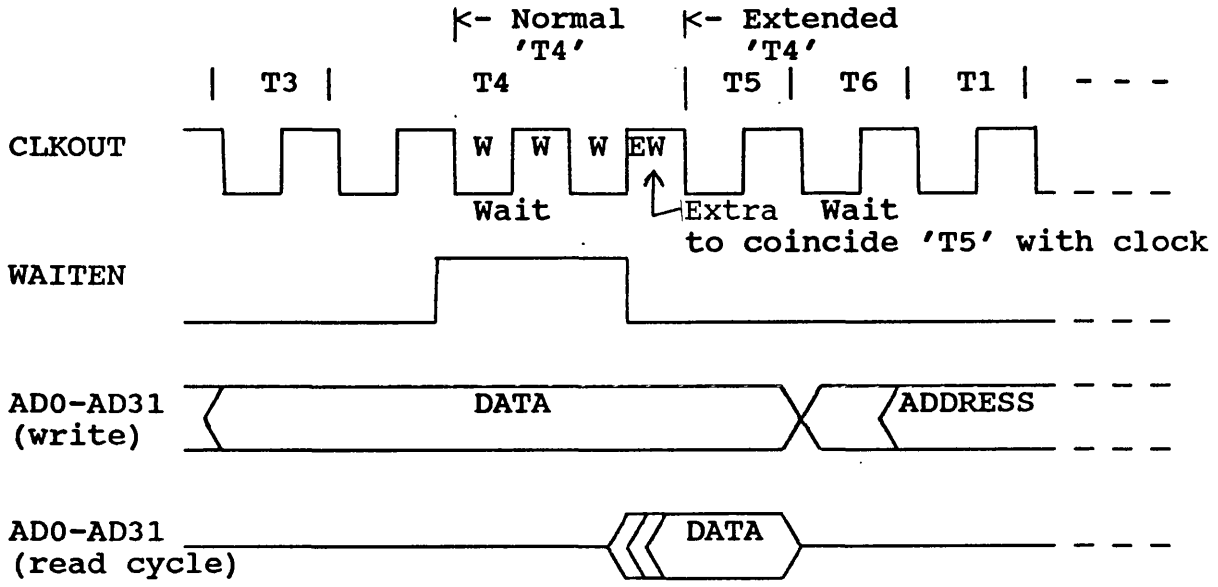


Fig 7.7 Timing diagram for the T800 with wait state.

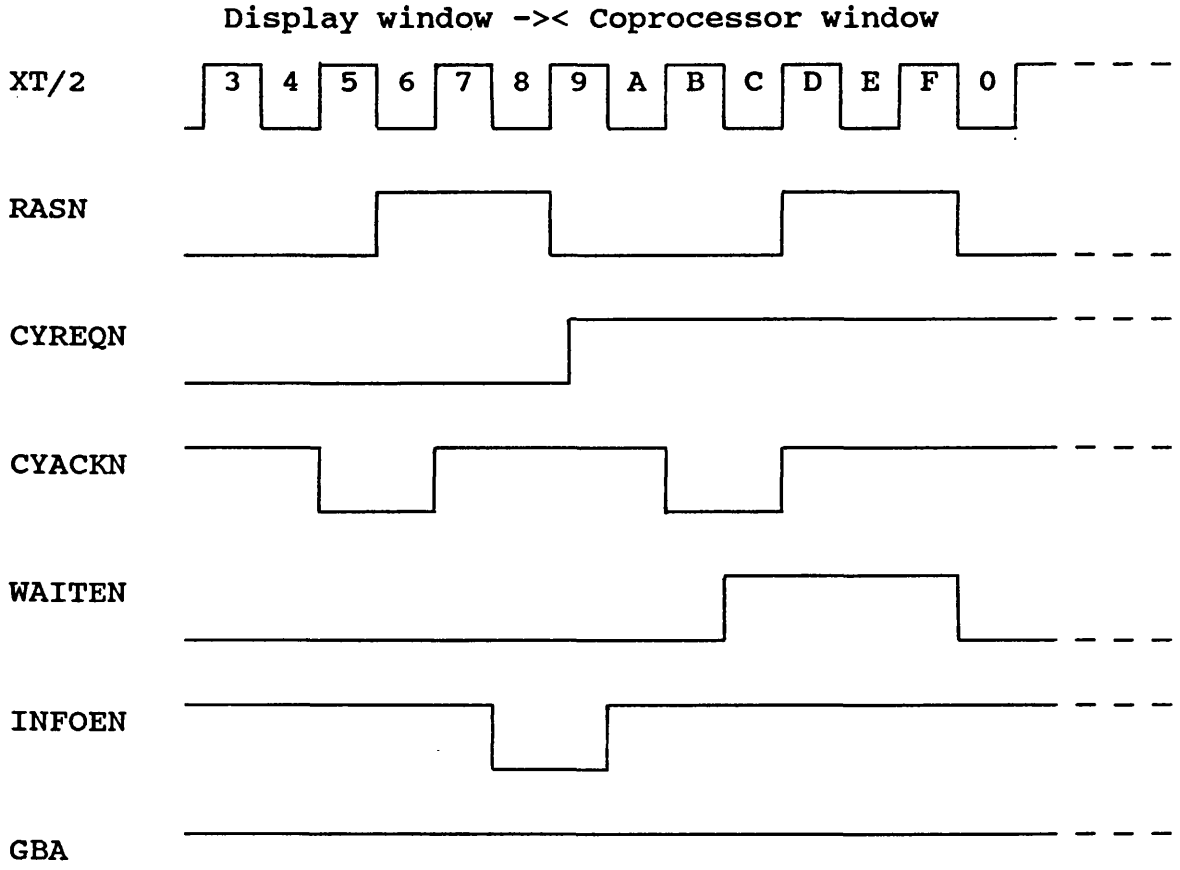


Fig 7.8 Timing diagram for coprocessor memory read cycle.

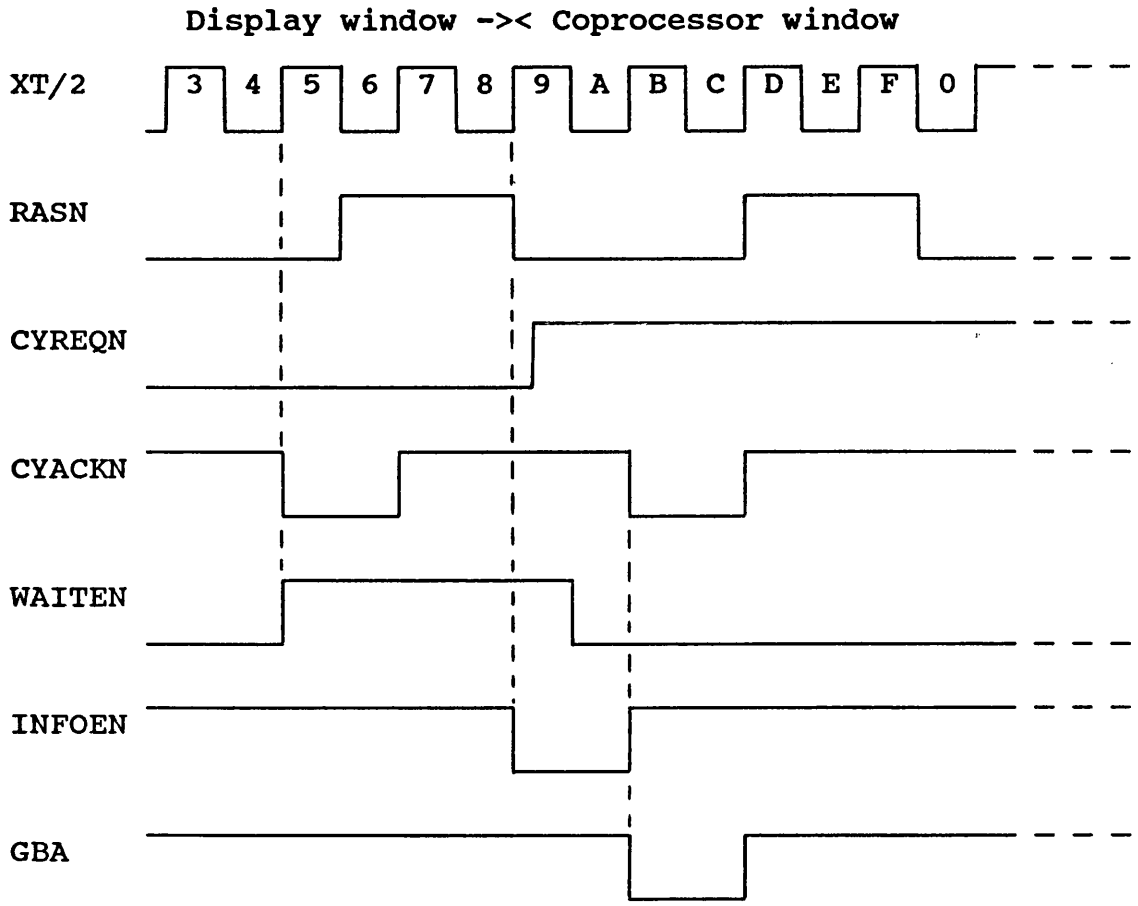
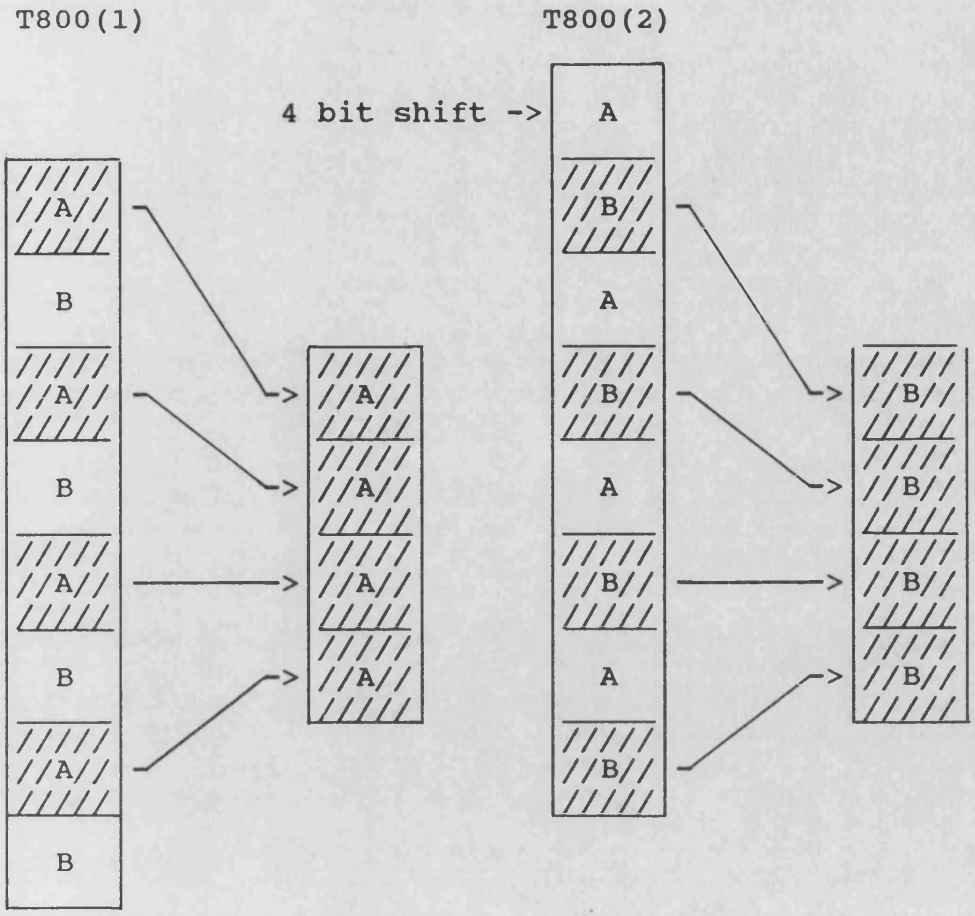


Fig 7.9 Timing diagram for coprocessor memory write cycle.



Nibble maping in two VSCs.

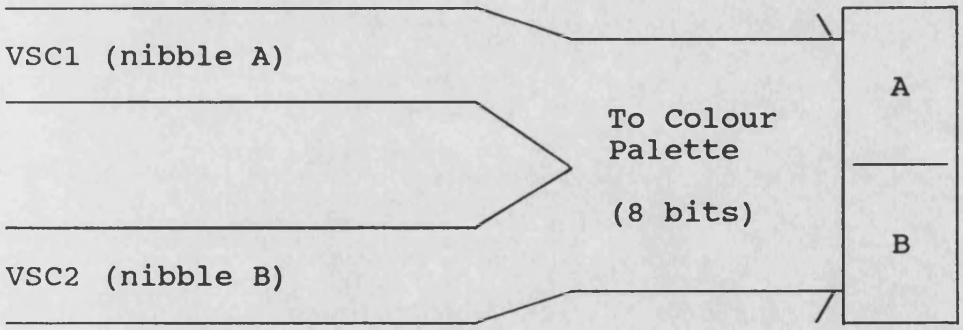


Fig 7.10 Two VSC system in Double Resolution mode.

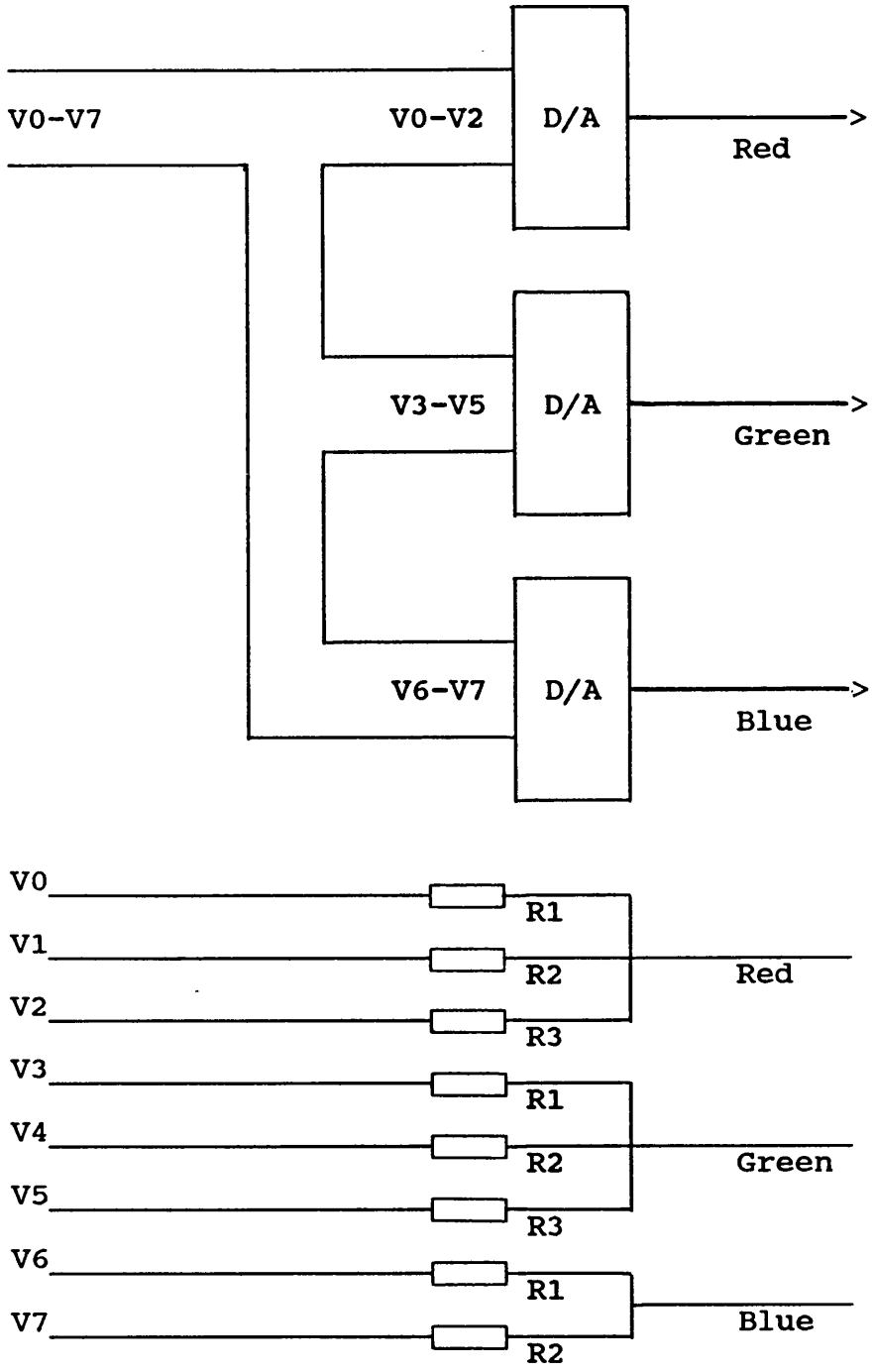


Fig 7.11 D/A convertor and Resistance network arrangements for generating weighted RGB output.

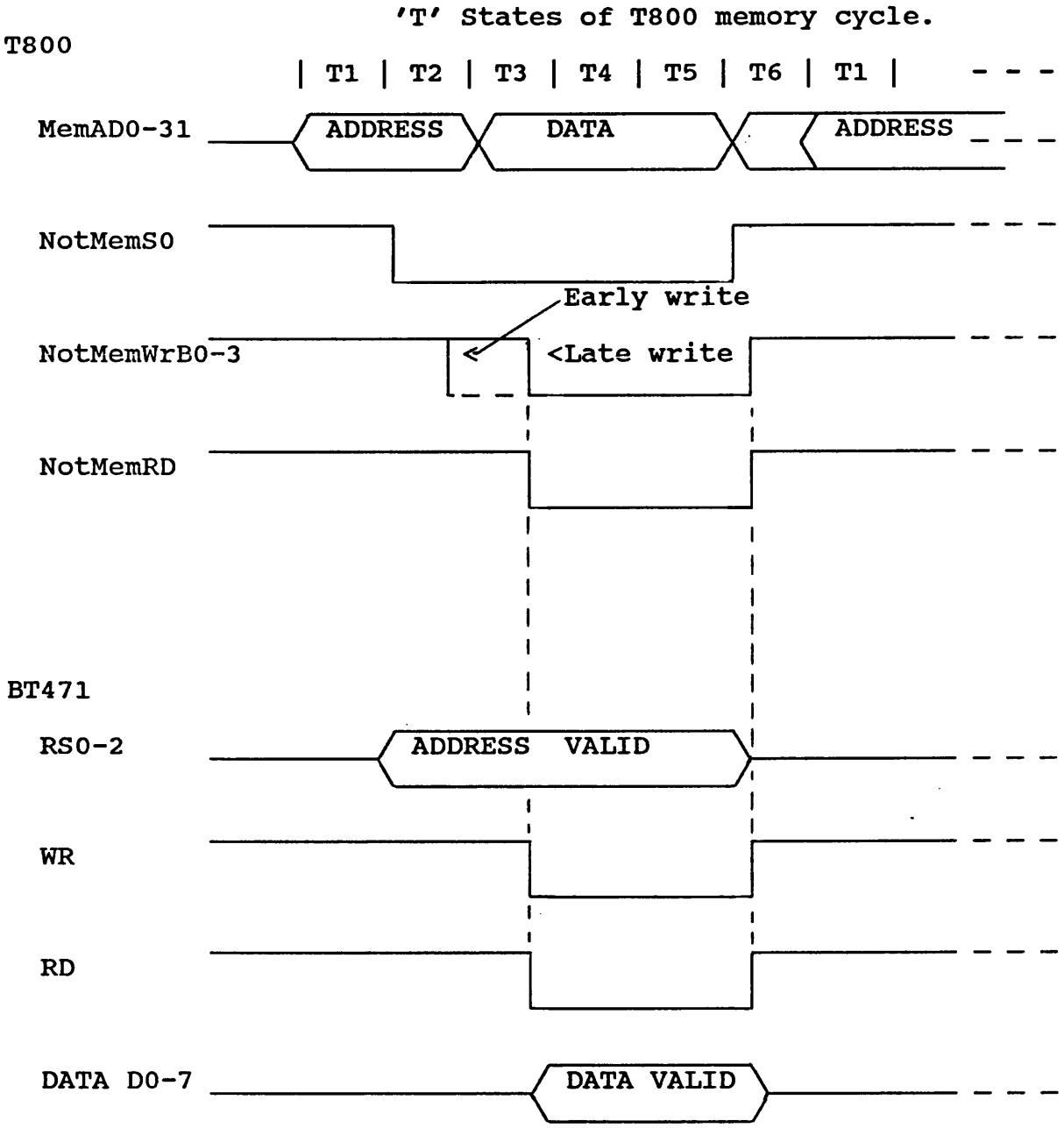


Fig 7.12 Timing diagram for BT471 interface.
T800 address is latched to make sure that
it remains valid when WR, RD become active.

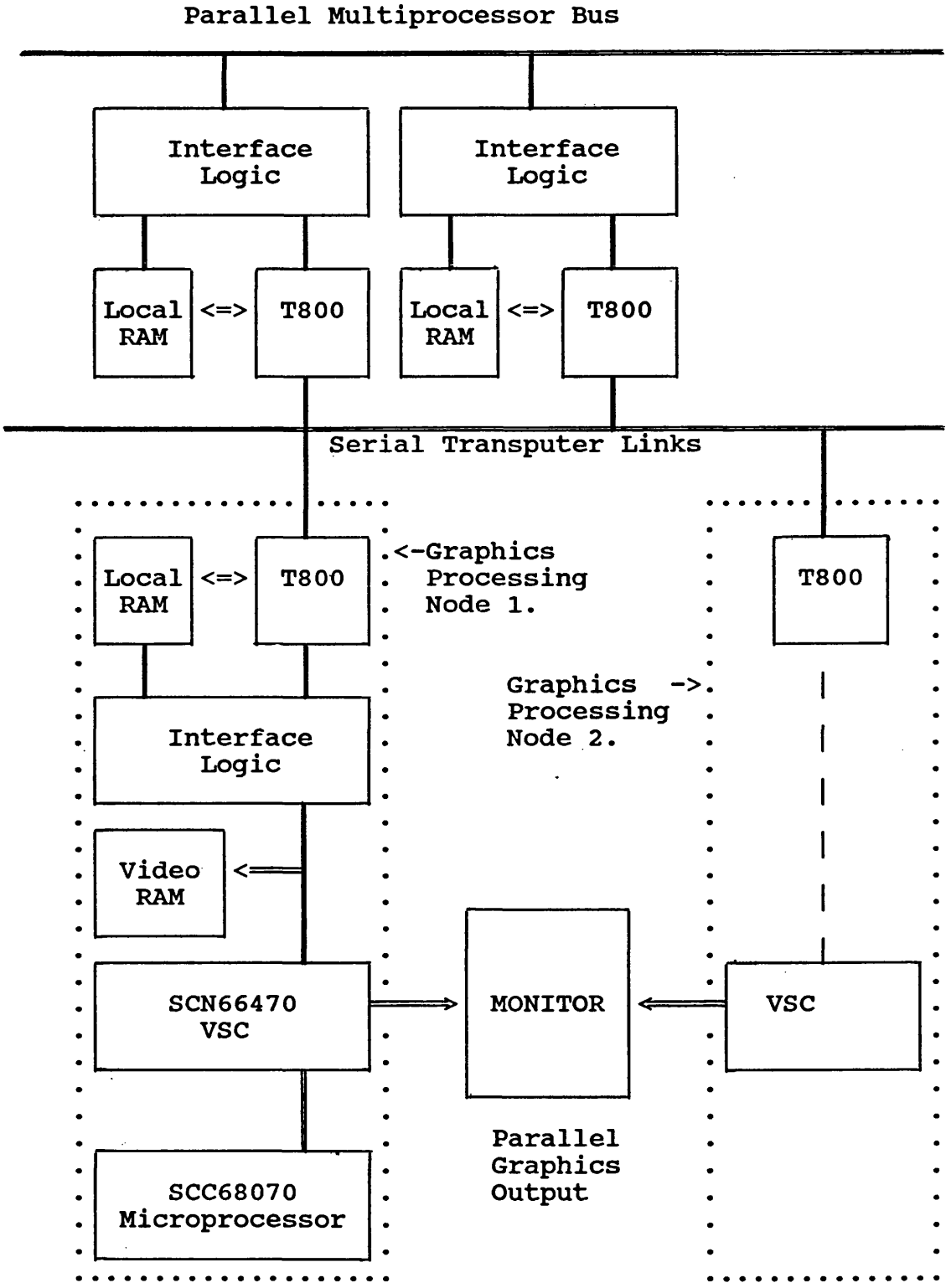


Fig 8.1 Combining the graphics processing nodes.

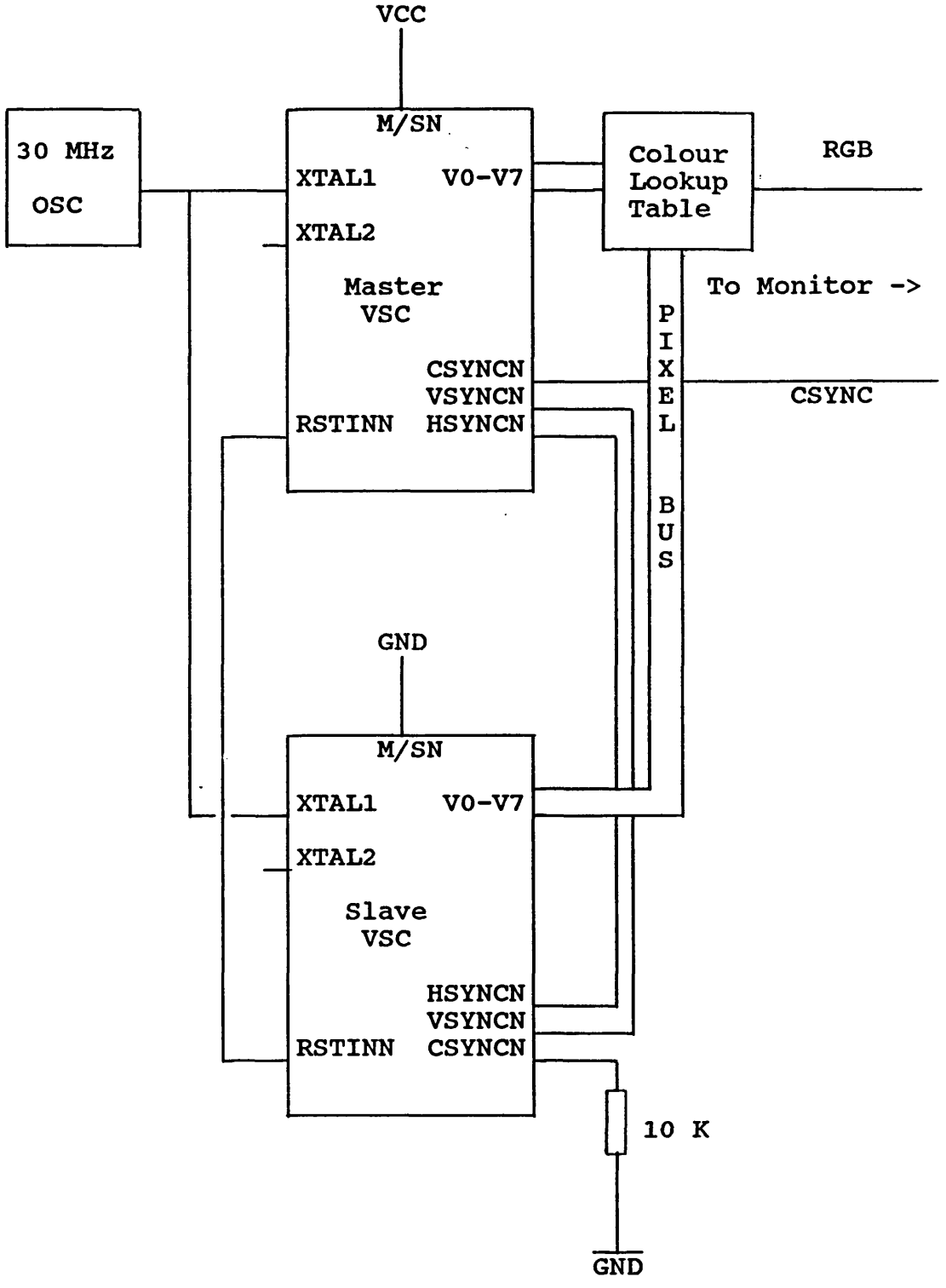


Fig 8.2 VSC master-slave operation.

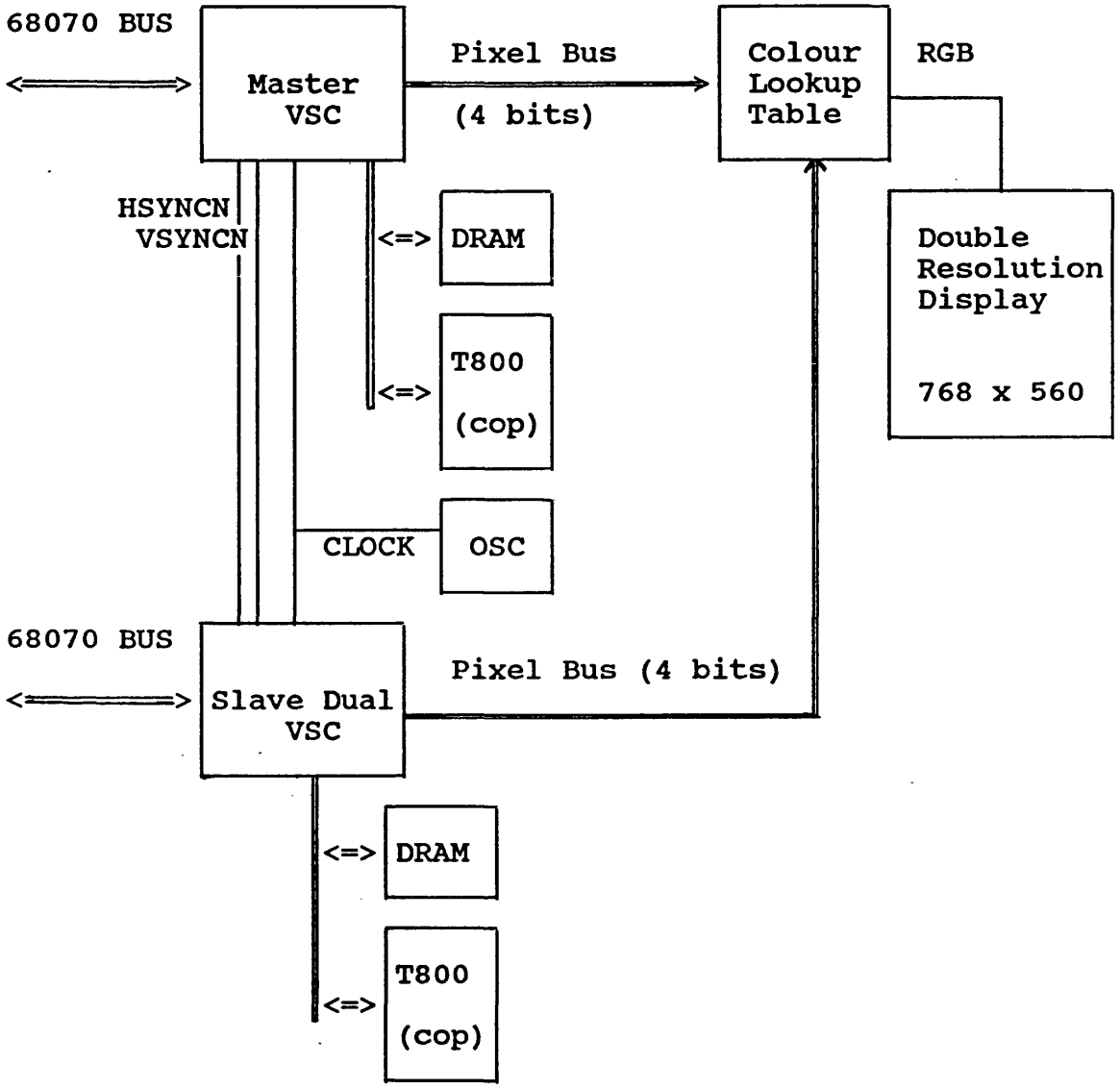


Fig 8.3 Two VSCs with parallel outputs. 4 bits per pixel from individual VSCs are combined to make 8 bits per pixel output.

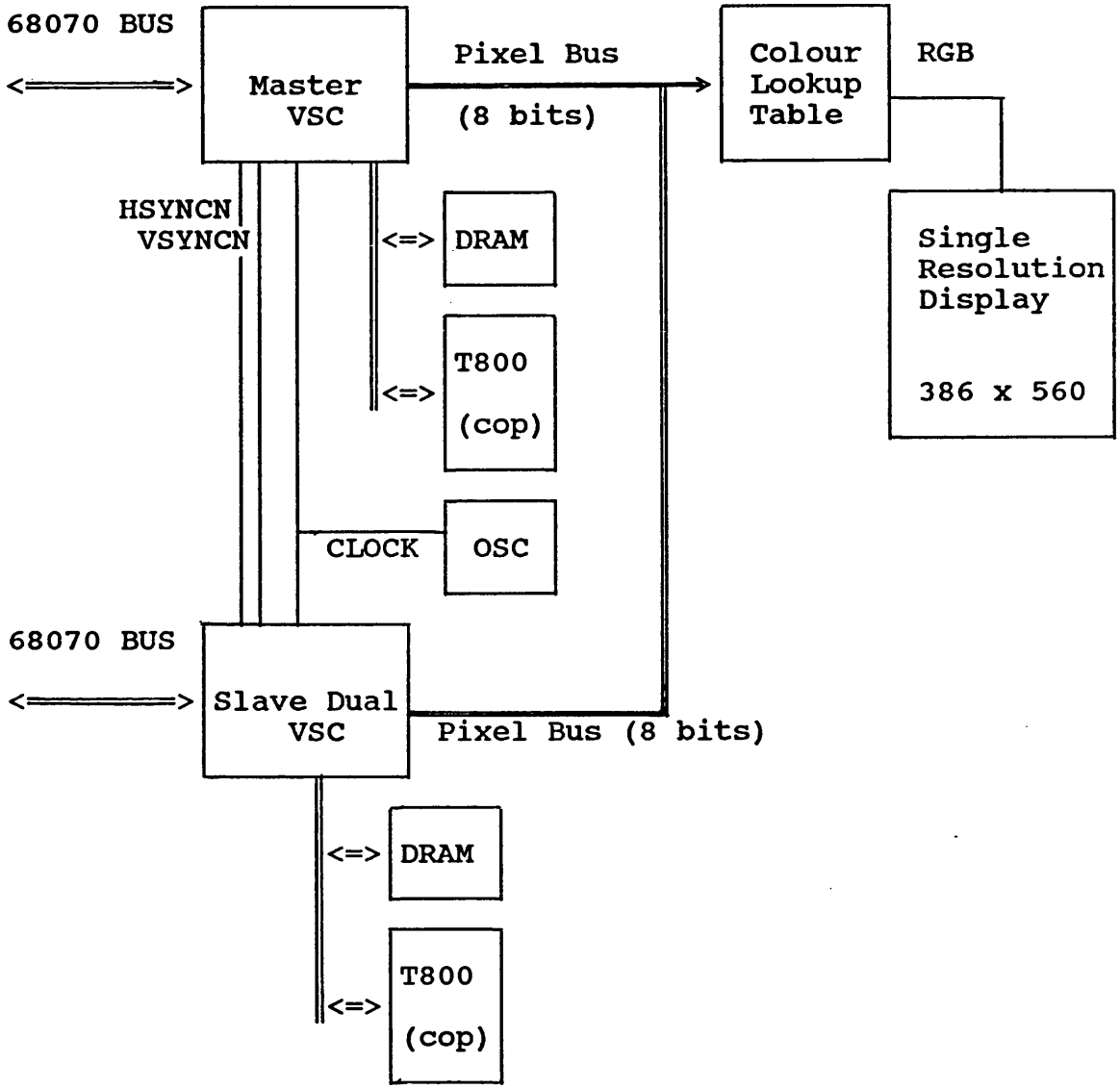


Fig 8.4 Two VSCs putting pixel information on the same bus. The output of one VSC is tristated in order to let the second VSC contribute that pixel.

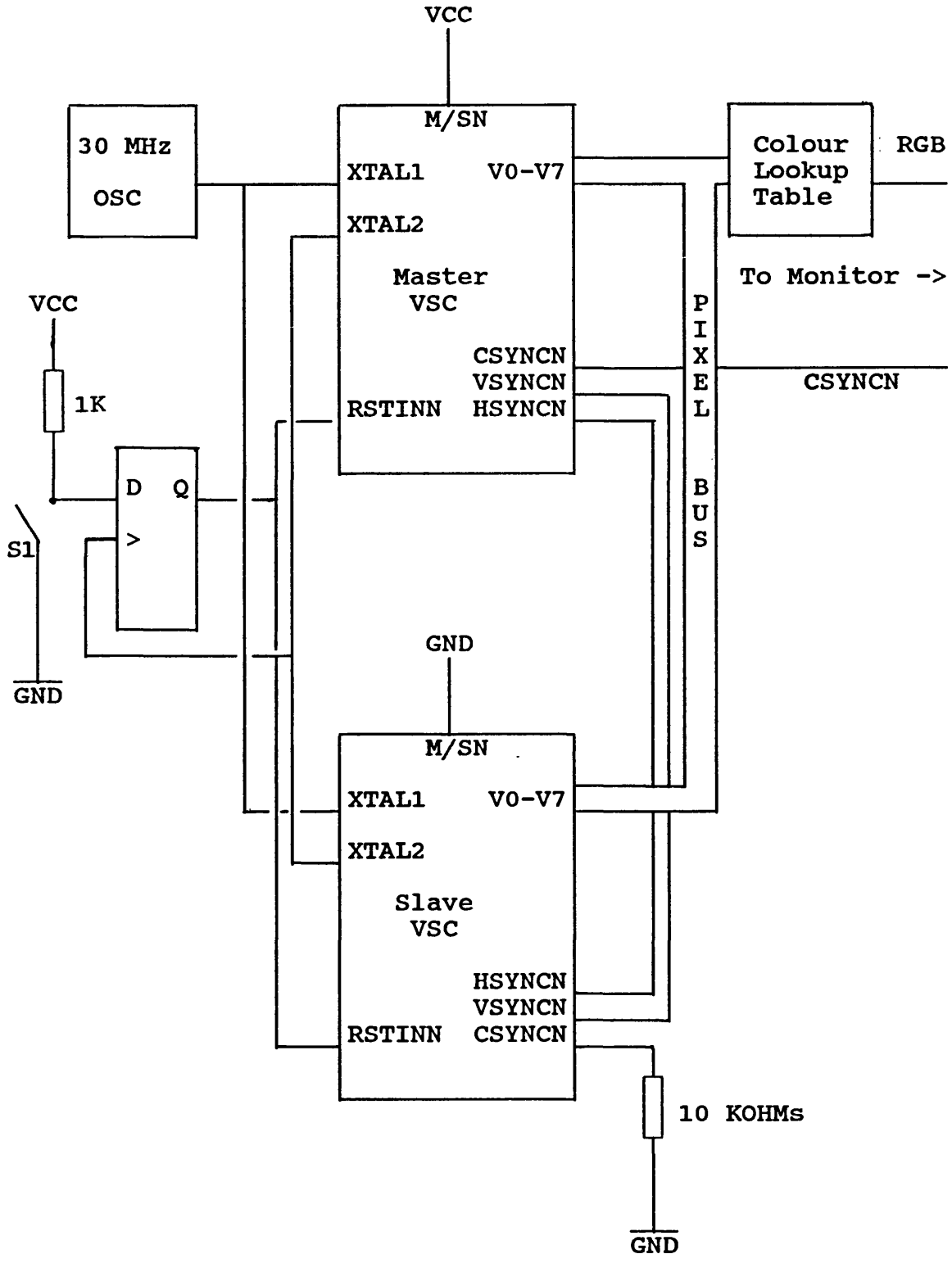


Fig 8.5 Reset arrangement for master/slave VSC operation.

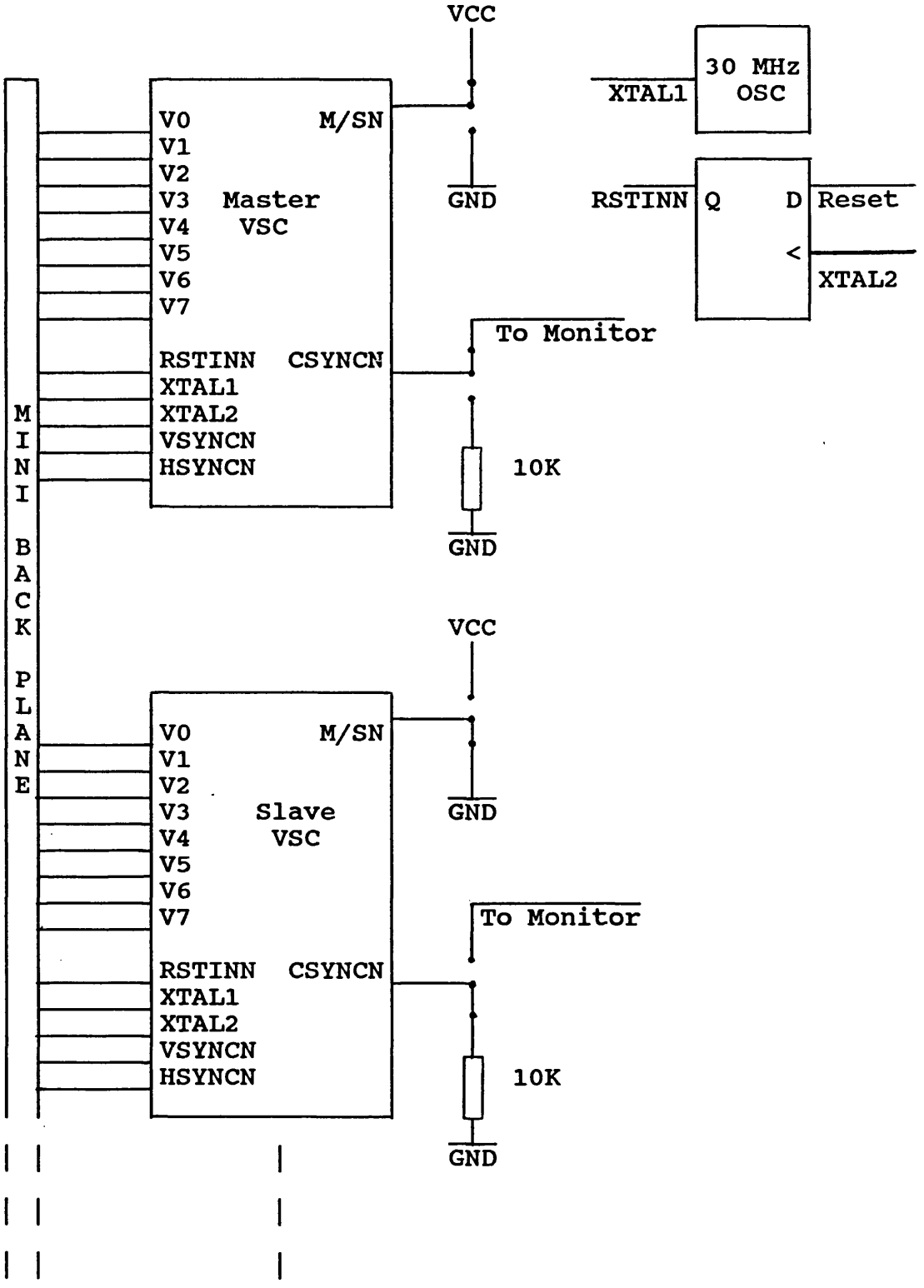
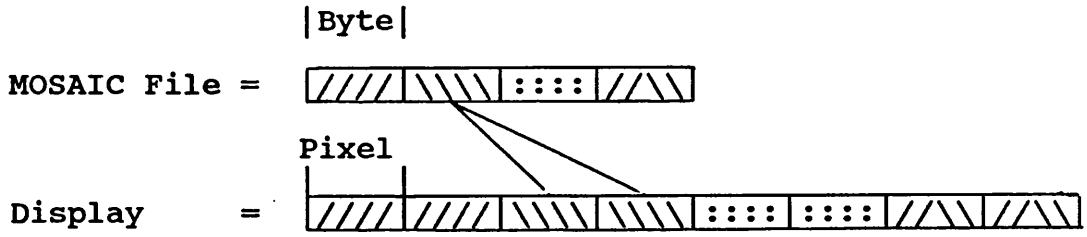
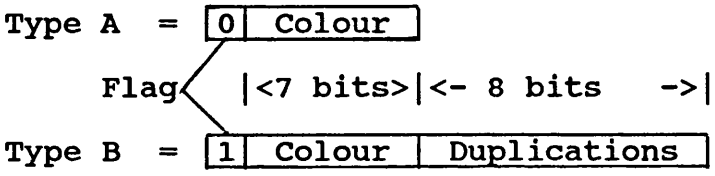


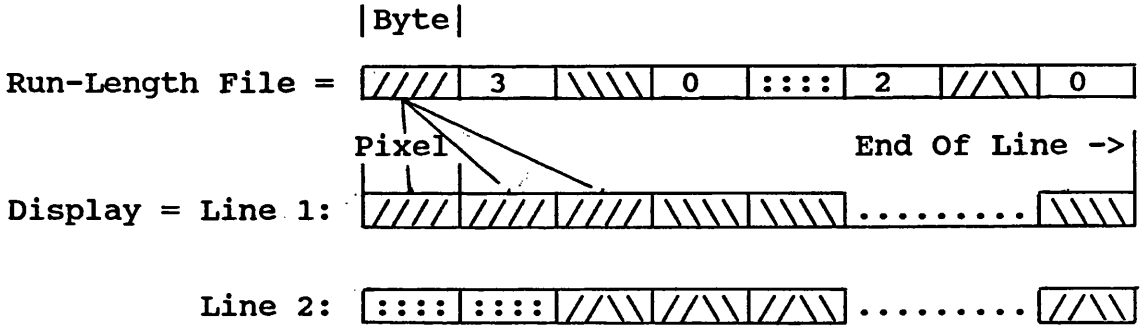
Fig 8.6 Master and Slave VSCs with mini Back Plane.



MOSAIC file display in 8 bit per pixel mode.
(Horizontal MOSAIC factor = 2)

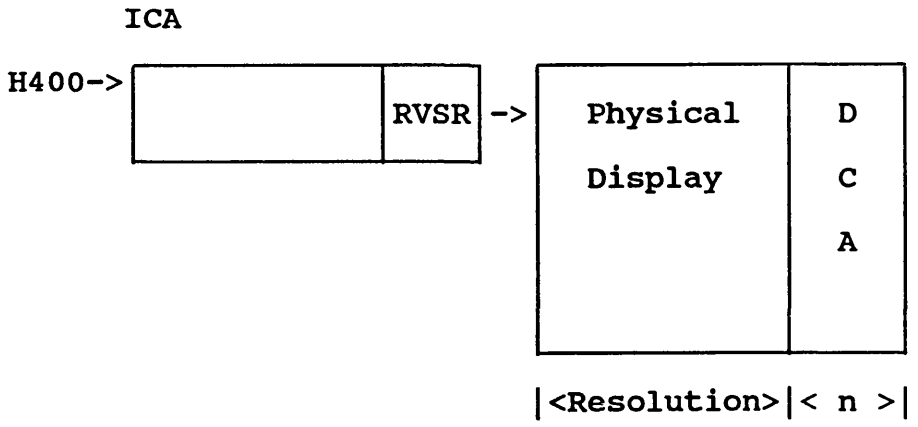


Run-Length file format in 8 bit per pixel mode.

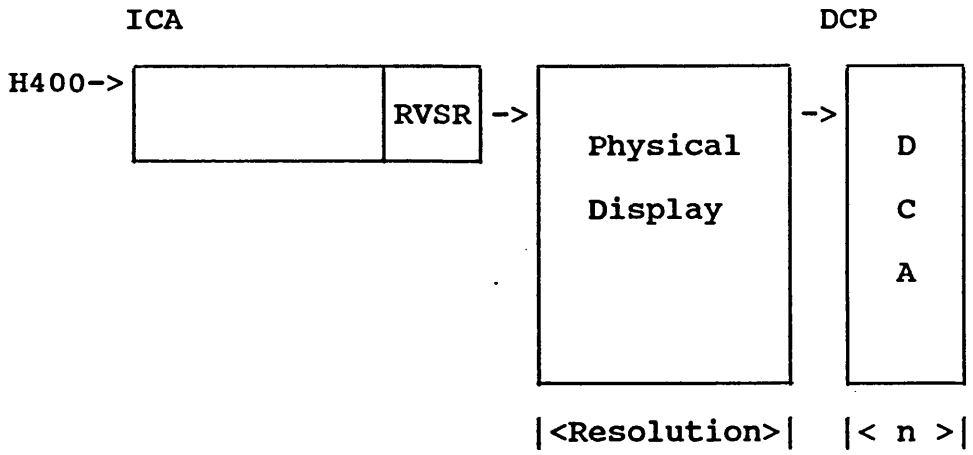


Run-Length file display in 8 bit per pixel mode.

Fig 8.7 MOSAIC and Run-Length files with their respective video displays.



Interleaved DCA and Physical display.



Independent DCA and Physical display.

RVSR = Reload VSR.
DCP = DCA Pointer.
n = 64 bytes or 16 bytes (Reduced DCA).

Fig 8.8 Interleaved DCA and Independent DCA mechanism.

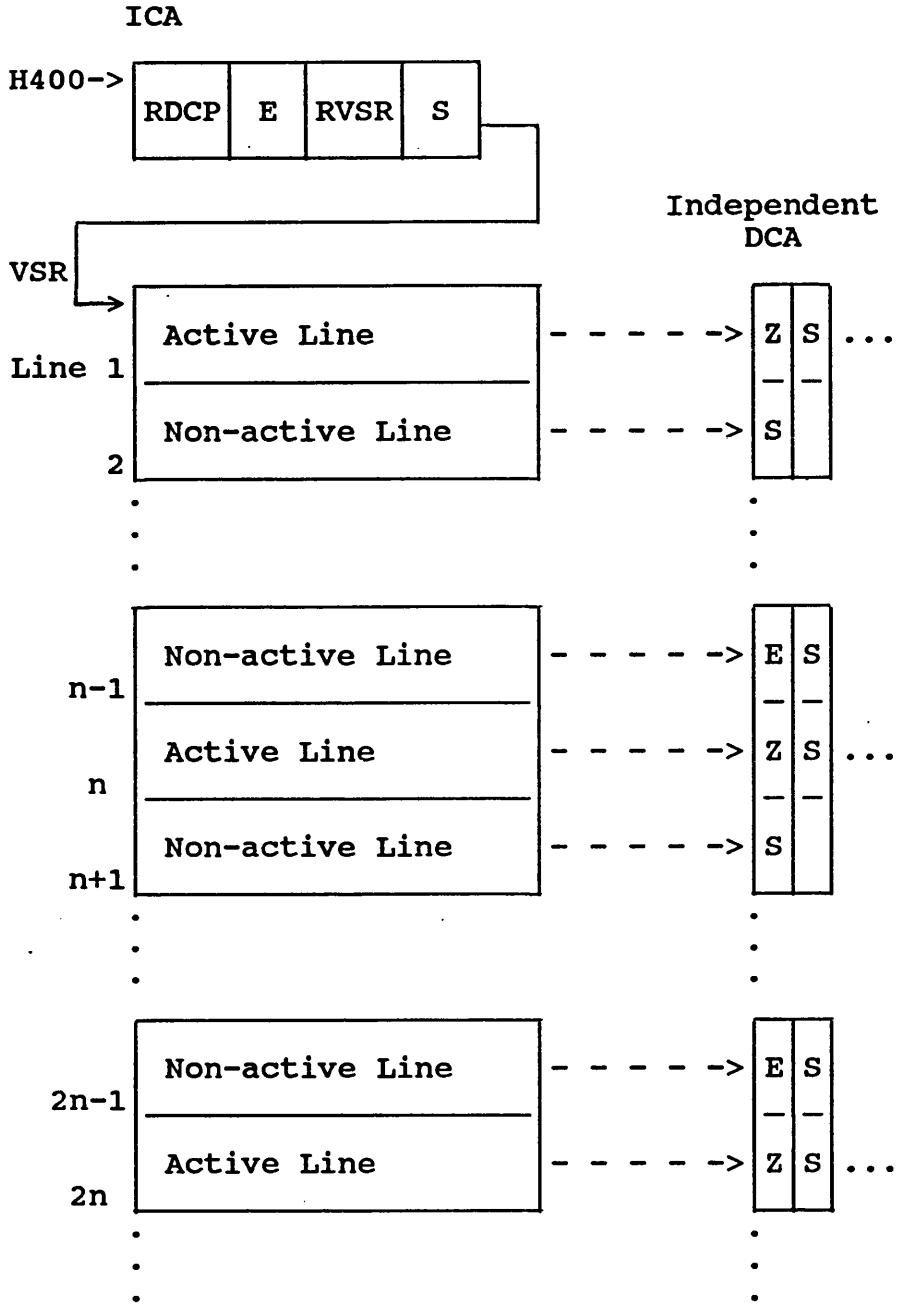


Fig 8.9 Bitmap file organisation for multiple VSC display system.

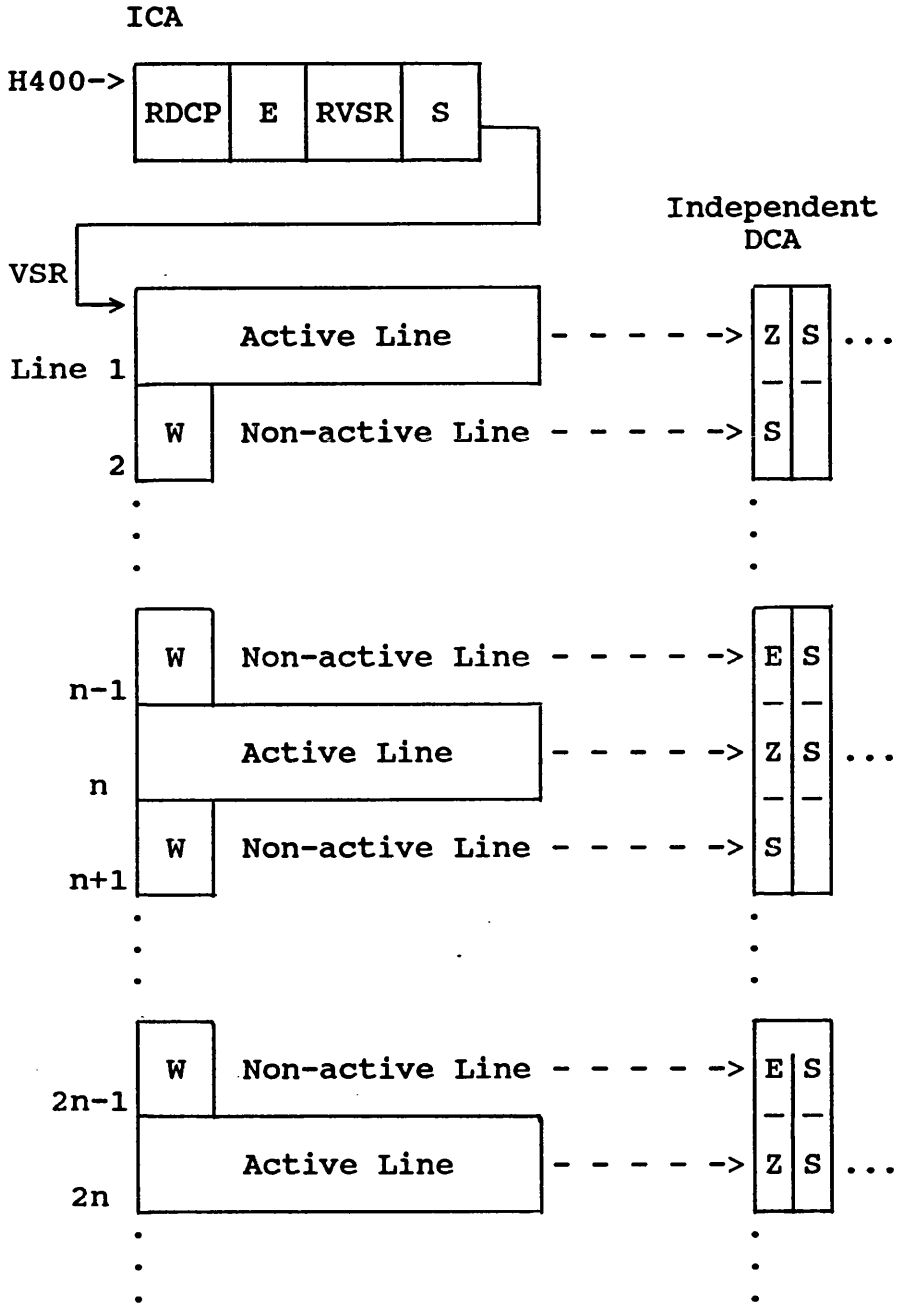


Fig 8.10 An alternative display file for multiple VSC system using both bitmap and run-length file organisations.